



# BXML Reference

Version 3.1.4

Thu Mar 30 11:57:05 CEST 2006

## Table of Contents

---

BXML B Tags	3
b:backbase	3
b:buffer	3
b:bxml	3
b:deck	4
b:infobox	4
b:menubar	5
b:menucell	6
b:menupane	6
b:menupopup	7
b:menurow	7
b:panel	7
b:panelset	8
b:slider	11
b:slider-option	12
b:treelist	12
b:treelistcell	13
b:treelistchildren	13
b:treelistrow	13
b:xhtml	14
BXML S Tags	15
s:apply-templates	15
s:attribute	15
s:attribute	15
s:behavior	15
s:bookmark	16
s:call-template	17
s:choose	17
s:controlpath	17
s:copy-of	18
s:copy-of	18
s:default	18
s:element	19
s:event	19
s:execute	20
s:for-each	20
s:fxstyle	21
s:gfx	26
s:gfxset	26
s:history	27
s:htmlstructure	27
s:httpcode	28
s:httpheader	29
s:if	29
s:include	29
s:initatt	30
s:innercontent	30
s:keys	30
s:loading	31
s:lock	32
s:otherwise	33
s:parallel	33

s:render	34
s:script	36
s:sequential	37
s:setatt	37
s:setstyle	37
s:state	47
s:stylesheet	48
s:super	48
s:task	49
s:tasklist	49
s:template	49
s:textnode	50
s:tooltip	50
s:value-of	50
s:value-of	51
s:variable	51
s:when	52
s:whenactive	53
s:with	53
s:xml	54
HTML Tag Extensions	55
xhtml:body	55
xhtml:form	55
xhtml:link	56
xhtml:meta	56
xhtml:script	56
xhtml:style	57
xhtml:td	57
xhtml:xmp	57
BXML Controls	59
b:accordion	59
b:accordionbody	59
b:accordionhead	59
b:barchart	59
b:barchart-bar	60
b:barchart-horizontal-values	60
b:barchart-series	61
b:barchart-value	61
b:box	61
b:button	61
b:combobox	62
b:combo-option	62
b:contextmenu	63
b:contextmenurow	63
b:datagrid	64
b:datagridbody	69
b:datagridhead	69
b:datagridheadcell	69
b:datagridrow	69
b:datatype	69
b:datatypes	69
b:datepicker	70
b:detailviewer	71
b:flash	71

b:linechart	72
b:linechart-horizontal-values	72
b:linechart-point	73
b:linechart-series	73
b:linechart-value	73
b:listgrid	73
b:listgridbody	78
b:listgridhead	78
b:listgridheadcell	78
b:listgridrow	78
b:livegrid	78
b:livegrid-listener	79
b:modal	79
b:modalbody	80
b:modalhead	80
b:navbody	80
b:navbox	80
b:navhead	81
b:navpanel	81
b:navpanelbody	82
b:navpanelhead	82
b:option	82
b:orgchart	82
b:pager	83
b:pager-item	84
b:pager-numbers	84
b:property	84
b:select	84
b:separator	85
b:sidebar	85
b:sidebarbox	85
b:spacer	86
b:spinner	86
b:statusbar	87
b:subordinate	87
b:tab	87
b:tabbox	88
b:taskbar	88
b:tilelist	89
b:tilelist-body	90
b:toolbar	90
b:toolbaritem	91
b:tree	91
b:window	92
b:windowarea	92
b:windowbody	93
b:windowhead	93
Generic Attributes	94
b:action	94
b:behavior	94
b:columnmode	94
b:cursor	94
b:disabled	95
b:drag	95

b:dragconstraint	96
b:dragconstrainttype	96
b:draggroup	96
b:dragmode	97
b:dragreceive	97
b:eventblock	98
b:focusgroup	98
b:focusindicator	99
b:focusitem	99
b:focusroot	99
b:followstate	99
b:lineconstraint	100
b:maxheight	100
b:maxwidth	100
b:minheight	101
b:minwidth	101
b:observe	101
b:resize	101
b:resizeconstraint	102
b:resizemode	102
b:singular	102
b:state	103
b:test	103
b:textselect	103
b:tooltiptext	103
b:zsort	103
BXML Commands	104
addclass	104
alert	104
assign	104
backward	105
blur	106
bookmark	106
bufferdirty	107
bxml2string	107
copy	108
deselect	109
disable	109
enable	110
execute	110
focus	110
focusgroup-next	111
focusgroup-previous	111
focusitem-next	112
focusitem-previous	112
focusroot-next	113
focusroot-previous	113
formxml	113
forward	114
fxtile	114
hide	115
html2string	115
invisible	116
js	116

load	116
maximize	117
move	118
movedown	119
moveup	119
msg	120
next	120
opacity	120
position	121
previous	123
print	123
remove	123
removeclass	124
render	124
resize	125
restore	125
resync	126
scrollto	127
select	127
select-deselect	128
send	129
set	129
setcookie	130
setpanelset	131
setstatus	132
setstruct	132
settext	132
setttitle	132
show	133
show-hide	133
sort	133
store	135
string2xml	135
submit	136
tile	136
transform	137
trigger	138
visible	139
visible-invisible	140
xml2string	140
xsl-transform	141
zsort	142
BXML Events	143
active	143
blocked-mousedown	143
blur	143
change	144
child-lost	144
click	145
close	145
command	145
construct	146
copy	146
dblclick	147

deselect	147
disable	147
drag-deeper	148
drag-drop	148
drag-enter	148
drag-leave	148
drag-receive	149
drag-reenter	149
drag-start	149
enable	149
focus	150
inactive	151
keydown	151
keyup	151
load	151
mousedeeper	152
mousedown	152
mouseenter	153
mouseleave	153
mousereenter	154
mouseup	155
open	155
receive	155
remove	156
resize	156
resizewindow	156
rmbclick	156
rmbdown	157
rmbup	157
scrollwindow	157
select	157
slide	158
submit	158
Backbase JavaScript Functions	159
bpc.addClass	159
bpc.checkEventBlock	159
bpc.execute	159
bpc.focus	160
bpc.forceDisplay	160
bpc.getBXMLAttribute	160
bpc.getCookie	161
bpc.getNextElementByNodeType	161
bpc.getNextElementByTagName	161
bpc.getPreviousElementByNodeType	161
bpc.getPreviousElementByTagName	162
bpc.getSize	162
bpc.getXpathVar	162
bpc.move	162
bpc.place	163
bpc.removeBXMLAttribute	163
bpc.removeClass	164
bpc.render	164
bpc.restoreDisplay	164
bpc.setBXMLAttribute	164

bpc.setCookie	165
bpc.setVariable	165
bpc.setXpathVar	166
bpc.task	166
bpc.toString	166
bpc.trigger	167
bpc.xpath	167
XPath Axes	167
ancestor	167
ancestor-or-self	167
attribute	168
child	168
descendant	168
descendant-or-self	168
following-sibling	168
object	168
parent	168
preceding-sibling	168
self	168
style	168
XPath Functions	169
abs	169
avg	169
boolean	169
bxml	169
ceiling	169
codepoints-to-string	169
compare	170
concat	170
contains	170
cookie	170
count	170
current	170
current-date	170
current-dateTime	171
current-time	171
day-from-date	171
day-from-datetime	171
days-from-duration	171
declared	171
deep-equal	171
distinct-values	171
empty	171
ends-with	172
error	172
escape-uri	172
exactly-one	172
exists	172
false	172
floor	172
hours-from-datetime	172
hours-from-duration	173
hours-from-time	173
html	173

id	173
implicit-timezone	173
index-of	173
insert-before	174
last	174
local-name	174
local-name-from-QName	174
lower-case	174
matches	174
max	174
min	174
minutes-from-dateTime	174
minutes-from-duration	175
minutes-from-time	175
month-from-date	175
month-from-dateTime	175
months-from-duration	175
name	175
node-name	175
normalize-space	175
not	175
number	176
one-or-more	176
position	176
remove	176
replace	176
reverse	176
root	176
round	176
round-half-to-even	176
seconds-from-dateTime	177
seconds-from-duration	177
seconds-from-time	177
starts-with	177
string	177
string-join	177
string-length	177
string-to-codepoints	177
subsequence	178
substring	178
substring-after	178
substring-before	178
sum	178
timezone-from-date	178
timezone-from-dateTime	178
timezone-from-time	178
tokenize	178
trace	179
translate	179
true	179
upper-case	179
xpath	179
year-from-date	179
year-from-dateTime	179

years-from-duration	179
zero-or-one	179
XPath Node Tests	179
node()	179
text()	180
XPath Variables	180
\$_currentPath	180
\$_dragCurrent	180
\$_dragParent	180
\$_dragTarget	180
\$_element	180
\$_eventSource	180
\$_httpObject	180
\$_mouseElement	180
\$_mouseSource	181
\$_selectedText	181
\$_source	181
\$_sourceIndex	181
\$_target	181
\$bpct_bookmark	181
\$bpct_browser	182
\$bpct_controlpath	182
\$bpct_dragMargin	182
\$bpct_dragOutline	182
\$bpct_dragSymbol	182
\$bpct_focusCurrent	183
\$bpct_focusCurrentElement	183
\$bpct_focusLastElement	183
\$bpct_fxTime	183
\$bpct_fxTimeout	184
\$bpct_hash	184
\$bpct_keyAlt	184
\$bpct_keyCtrl	184
\$bpct_keyShift	184
\$bpct_mouseX	184
\$bpct_mouseY	185
\$bpct_path	185
\$bpct_resizeLine	185
\$bpct_scrollLeft	185
\$bpct_scrollTop	185
\$bpct_tooltipClass	185
\$bpct_tooltipTimeout	186
\$bpct_version	186
\$bpct_windowHeight	186
\$bpct_windowWidth	186
\$bpct_xsltParser	186

Products and technologies mentioned in this document are trade marks or registered marks of their respective owners. Backbase may have patents, patent applications, trademarks, copyrights or other intellectual property rights covering the subject matter of this document. Backbase products include software developed by the Apache Software Foundation.



## BXML B Tags

### b:backbase

The `b:backbase` tag is a runtime tag that is generated automatically for each `xmp` tag defined in your source code (that has the attribute `b:backbase="true"` set) so that the content of the tags can be rendered. The `b:backbase` tag is actually a `div` element.

### b:buffer

Defines the child element of a deck control; the buffer does not load its contents, defined in a file specified by the `b:url` attribute, until it is selected for the first time. Buffered loading is useful because you can defer loading, reducing the initial loading time. By default, the first buffer in the deck is loaded, although you can explicitly specify the first buffer to load using the `b:state="selected"` attribute.

The file loaded by the buffer tag is appended to the BXML space, and does not replace the content.

Note you can use the `bufferdirty` command to force a reload of the file contents.

#### Attributes

---

##### [Required] `b:url`

Specify the path of the file to load, relative to the startup file of your Backbase application.

---

#### Parent Tags [Required]

##### `b:deck`

### Implementation

In the following example, the deck's child elements are loaded by the `b:buffer` tag. The `b:buffer` delays the loading of the file's contents until it is selected.

```
<b:deck id="buffer_spideck" style="border: solid 1px #959595; width: 400px; height:400px">
  <b:buffer b:url="data/seven.xml" />
  <b:buffer b:url="data/2001.xml" />
  <b:buffer b:url="data/bella.xml" />
</b:deck>
<b:button b:action="previous" b:target="id('buffer_spideck')">Previous</b:button>
<b:button b:action="next" b:target="id('buffer_spideck')">Next</b:button>
```

Where the contents of the loaded XML response files (in the `b:url` attribute) contain, for example:

```
<?xml version="1.0" ?>
<div xmlns="http://www.w3.org/1999/xhtml" xmlns:b="http://www.backbase.com/b"
      xmlns:s="http://www.backbase.com/s">
  <h3>2001: A Space Odyssey (1968)</h3>
  <p>An influential science fiction film directed by Stanley Kubrick...</p>
</div>
```

### b:bxml

Marks the beginning of a *BXML space*, inside a `b:xhtml` element. Child nodes of this element are processed by the BXML parser.

### Implementation

In the following example, there is a tooltip on the `span` element. The first tooltip field is not rendered, even though it is present in the source code, because it is contained by a `b:xhtml` tag, which signals the start of the XHTML space. In this XHTML space, the BXML parser is disabled and since `s:tooltip` is a BXML tag it cannot be rendered. Within such an XHTML space the BXML parser can be re-enabled by using the `b:bxml` tag, which sig-

nals the start of a BXML space within this XHTML space.

```
<b:xhtml>
  <p>Normal XHTML space: b: and s: prefixed tags are not parsed...
    <br />
    <span>Text.
      <s:tooltip>Tooltip text.</s:tooltip>
    </span>
  </p>
  <b:bxml>
    <p>Normal BXML space: b: and s: prefixed tags are parsed...
      <span>Text.
        <s:tooltip>Tooltip text.</s:tooltip>
      </span>
    </p>
  </b:bxml>
</b:xhtml>
```

## b:deck

A multi-paged control in which only one of the pages is visible at a time. Comparable to a deck of cards, you can use it to create wizard-like and tabbed interfaces. Each child tag represents a page in the deck, which can be cycled through by issuing the `next` and `previous` commands. You can define its children using, for example, `div` elements or `b:buffer` to load contents from a file.

### Child Tags

You can define any BXML (and HTML) elements as children of this tag. You can also use the following tag-specific child tags:

`b:buffer`

### Implementation

In the following example, a simple deck control with static content is defined including "Previous" and "Next" links that allow you to navigate through the deck's child elements.

```
<b:deck>
  <div>1 This is a deck...</div>
  <div>2...a control to create...</div>
  <div>3...multi-paged documents.</div>
</b:deck>
<a b:action="previous" b:target="#/b:deck">&lt; Previous</a>
<a b:action="next" b:target="#/b:deck">Next &gt;</a>
```

In the following example, the deck's child elements are loaded by the `b:buffer` tag. The `b:buffer` delays the loading of the file's contents until it is selected.

```
<b:deck id="buffer_spideck" style="border: solid 1px #959595; width: 400px; height:400px">
  <b:buffer b:url="data/seven.xml" />
  <b:buffer b:url="data/2001.xml" />
  <b:buffer b:url="data/bella.xml" />
</b:deck>
<b:button b:action="previous" b:target="id('buffer_spideck')">Previous</b:button>
<b:button b:action="next" b:target="id('buffer_spideck')">Next</b:button>
```

Where the contents of the loaded XML response files (in the `b:url` attribute) contain, for example:

```
<?xml version="1.0" ?>
<div xmlns="http://www.w3.org/1999/xhtml" xmlns:b="http://www.backbase.com/b"
  xmlns:s="http://www.backbase.com/s">
  <h3>2001: A Space Odyssey (1968)</h3>
  <p>An influential science fiction film directed by Stanley Kubrick...</p>
</div>
```

## b:infobox

An `infobox` provides additional content to the user that and stays visible while you inter-

act with its content; clicking outside the infobox hides the infobox. Open an infobox by **selecting** it using the `select` command.

## Implementation

In the following example, clicking the link opens the infobox control. This infobox contains a second infobox which has a link to return to the first infobox. Clicking anywhere outside the infobox closes the control.

```
<p>
    <a b:action="select" b:target="id('tre_infobox1')">Click to open infobox.</a>
</p>
<b:infobox id="tre_infobox1">
    <h1>This is the first infobox</h1>
    <p>The infobox allows user interaction and can contain links</p>
    <p>
        <a b:action="select" b:target="id('tre_infobox2')">Link to second infobox</a>
    </p>
</b:infobox>
<b:infobox id="tre_infobox2">
    <h1>This is the second infobox</h1>
    <p>You can click through a whole series of infoboxes. They will stay open until you have
    clicked outside of the infobox</p>
    <p>
        <a b:action="select" b:target="id('tre_infobox1')">Back to the first infobox</a>
    </p>
</b:infobox>
```

## b:menubar

A horizontally oriented **menu**. The **menubar** is the main container bar for menus, much like the top-most collection of menu items (file, edit, view and so on), found in most desktop applications. The `b:menubar` tag may only contain `b:menurow` tags, which will be laid out horizontally.

This control is a core control implemented in the BPC. If you want to use `menubar` with styling applied, you must add an `s:include` to your startup file and specify the `b:url` attribute to the control file, for example, `/controls/backbase/b-menubar/b-menubar.xml` directory.

### Child Tags

[Required] `b:menurow`

### Attributes

`b:mode`

Define the positioning of a menu when it is opened.

#### Values:

- after-end
- after-pointer
- after-start
- at-pointer
- before-end
- before-start
- end-after
- end-before
- overlap
- start-after
- start-before

`b:trigger`

Specify when a menu is opened; `over` when the cursor goes over the menu, or `click` when you click the menu (the default).

#### Values:

- click
- over

`b:stayopen`

Causes the `b:menubar` to stay open once activated (autoclose disabled). The default is `false`.

## Implementation

In the following example, the `b:menubar` has two main menu items. Select the File > Export menu item to open a nested menu.

```
<s:include b:url="/Backbase/controls/backbase/b-menubar/b-menubar.xml" />
<div style="width: 100%; height: 100%; position: relative">
    <div style="border-color: #b8ced7; border-width: 1px 1px 0 1px; border-style: solid; height:23px;overflow:hidden;">
        <b:menubar>
            <b:menurow b:label="File">
                <b:menupopup>
                    <b:menupane>
                        <b:menurow>
                            <b:menucell>Open</b:menucell>
                        </b:menurow>
                        <b:menurow>
                            <b:menucell>Save</b:menucell>
                        </b:menurow>
                        <b:menurow>
                            <b:menucell>Export</b:menucell>
                            <b:menupopup>
                                <b:menupane>
                                    <b:menurow>
                                        <b:menucell>To gif</b:menucell>
                                    </b:menurow>
                                    <b:menurow>
                                        <b:menucell>To png</b:menucell>
                                    </b:menurow>
                                </b:menupane>
                            </b:menupopup>
                        </b:menurow>
                        <b:menurow>
                            <b:menucell>Save as ...</b:menucell>
                        </b:menurow>
                    </b:menupane>
                </b:menupopup>
            </b:menurow>
            <b:menurow>
                <b:menucell>Edit</b:menucell>
                <b:menupopup>
                    <b:menupane>
                        <b:menurow>
                            <b:menucell>Copy</b:menucell>
                        </b:menurow>
                        <b:menurow>
                            <b:menucell>Cut</b:menucell>
                        </b:menurow>
                        <b:menurow>
                            <b:menucell>Paste</b:menucell>
                        </b:menurow>
                    </b:menupane>
                </b:menupopup>
            </b:menurow>
        </b:menubar>
    </div>
</div>
```

## **b:menucell**

The *menucell* is a column in a *menurow*. By using multiple menucell tags you can create placeholders for icons or images in your *menu*. For simple labels you can also use the `b:label` attribute on the `b:menurow` tag.

Parent Tags [Required]

`b:menurow`

## **b:menupane**

A vertically oriented *menu*. The *menupane* tag may only contain `b:menurow` tags. Each

row is displayed underneath each other.

#### Child Tags

[Required] b:menurow

#### Parent Tags

[Required] b:menupopup

## b:menupopup

*Menupopup* is the popup box that appears when you click on a *menurow* item in a *menubar* or in a nested menu. It should be placed inside a **b:menurow** tag, where it becomes the main containing element for a nested menu. Side-by-side with the **b:menupane** tag containing the actual nested menu, you can put any other HTML or BXML tags inside it, for example **div** elements to act as CSS hooks to change the appearance of the menu.

#### Attributes

##### b:mode

Define the positioning of a menu when it is opened.

Values:  
 after-end  
 after-pointer  
 after-start  
 at-pointer  
 before-end  
 before-start  
 end-after  
 end-before  
 overlap  
 start-after  
 start-before

##### b:trigger

Specify when a menu is opened; **over** when the cursor goes over the menu, or **click** when you click the menu (the default).

Values:  
 click  
 over

#### Child Tags

[Required] b:menupane

#### Parent Tags [Required]

b:menurow

## b:menurow

The *menurow* is a container element for *menucell* tags and optionally a *menupopup* tag. By adding one or more **b:menucell** tags in the menurow you can create columns (useful if, for instance, you want icons next to the text in the menu). Alternatively, if you do not need multiple columns, you can use the **b:label** attribute instead of the **b:menucell** tag. In order to create submenus, you can put a **b:menupopup** tag inside the menurow.

For *menurow*, you can also use **b:label** as a shorthand alternative for the required *menucell*.

#### Attributes

##### b:label

Specify a String value that displays text in a set location for a control.

#### Child Tags

b:menucell , b:menupopup

#### Parent Tags [Required]

b:menubar , b:menupane

## b:panel

The `b:panel` tag generates a context/space area inside of a panel set. It is within the `b:panel` tag that you are going to present the contents of your web page. This tag is always a child node of a `b:panelset` tag.

#### Child Tags

You can define any BXML (and HTML) elements as children of this tag.

#### Parent Tags [Required]

`b:panelset`

The `b:panelset` tag creates a container element which effectively subdivides a page/screen area in column and row oriented panels. It is at the level of a given `b:panelset` tag that you define the dimensions and column/row orientation of its `b:panel` child elements. A `b:panelset` tag can also contain other `b:panelset` node-sets.

A panelset creates an HTML structure which consists of a series of dimensional `div` elements. By default the panelset is set to absolute positioning and will occupy the entire area of its first relatively or absolutely positioned parent.

A panelset can be given either a horizontal layout by specifying the rows using `b:rows` or a vertical one by specifying the columns using `b:columns`. You can specify that a panel needs to take the remaining space by giving it a '\*' instead of a unit definition. Note that only one '\*' is allowed in a single 'rows' or 'columns' specification.

**Note:** Internet Explorer performs slower if ancestor elements have the style attribute `height: 100%`. To resolve this, set the parent elements, that have 100% height or 100% width, to `overflow: auto` or `overflow: hidden`.

#### Attributes

##### `b:rows`

Specify row dimensions defining values or a space separated set of values in %, px, pc, pt, em, ex, in, cm or mm. You can also use the wildcard asterisk (\*) sign to fill the remaining space, in which case all values before or after the wildcard must use the same unit of measurement. For example, "50px 100px \*".

##### `b:cols`

Specify column dimensions defining values or a space separated set of values in %, px, pc, pt, em, ex, in, cm or mm. You can also use the wildcard asterisk (\*) sign to fill the remaining space, in which case all values before or after the wildcard must use the same unit of measurement. For example, "50px 100px \*".

#### Child Tags [Required]

`b:panel` , `b:panelset`

## Implementation

A panelset wraps itself to the size of its containing element. If you want to use the panelset to divide your entire window area, set the parent html, body and other tags of the panelset to a height of 100%. For example:

The following example shows a basic panelset:

```
<style type="text/css">
#panel_left {
    border-width: 0px 5px 0px 0px;
    border-style: solid;
    border-color: ButtonFace;
}
```

```
#panel_top {
    border-width: 0px 0px 5px 0px;
    border-style: solid;
    border-color: ButtonFace;
    overflow: hidden;
}
#panel_bottom {
    overflow: hidden;
}</style>

<xmp b:backbase="true" style="height:100%;">
<b:panelset b:cols="25% *">
    <b:panel id="panel_left" b:resize="e" b:minwidth="30px" b:resizeconstraint=". . .">
        <p>Add content here</p>
    </b:panel>
    <b:panelset b:rows="40% *">
        <b:panel id="panel_top" b:resize="s" b:minheight="30px" b:resizeconstraint=". . .">
            <p>Add content here</p>
        </b:panel>
        <b:panel id="panel_bottom">
            <p>Add content here</p>
        </b:panel>
    </b:panelset>
</b:panelset>
</xmp>
```

The following example is a more complex panelset. The panelset uses cookies to store the size of the panels when it is resized. The panels also display scrollbars when the content of the panel is larger than its parent.

The first code example defines a JavaScript function that is used to get an id for the current page based on its url (the function is called from XPath), the second defines the CSS classes used by the panelset, and the third the BXML event handers, behaviors and panelset.

```
<script type="text/javascript">
    //
    function idpage(){
        return [("page" + location.protocol + location.host +
location.pathname).match(/\w/g).join("")];
    }
//</script>
```

```
<style type="text/css">
body {
    background-color: Window;
    overflow: hidden;
}
#panel_left {
    border-width: 0px 5px 0px 0px;
    border-style: solid;
    border-color: ButtonFace;
}
#panel_top {
    border-width: 0px 0px 5px 0px;
    border-style: solid;
    border-color: ButtonFace;
    overflow: hidden !important;
}
#panel_bottom {
    overflow: hidden !important;
}
.content_wrapper {
```

```

        width: 100%;
        height: 100%;
        border-width: 1px;
        border-style: solid;
        border-color: ButtonShadow ButtonHighlight ButtonHighlight ButtonShadow;
        padding: 15px;
        overflow: auto;
        color: WindowText;
    }</style>

```

The following code has:

- A `construct` event handler that sets the contents of the CSS class of the resize line
- A `construct` event handler that sets the panelset rows to match the cookie value, and tests to see whether the cookie exists (in case of first time user, or cookies have been removed or disabled)
- A `resize` event handler that sets the value of the cookie when the panelset is resized
- A `construct` event handler that sets the panelset columns to match the cookie value, and tests to see whether the cookie exists (in case of first time user, or cookies have been removed or disabled)

```

<xmp b:backbase="true" style="height:100%;">
    <s:event b:on="construct">
        <s:task b:action="assign" b:target="$bpc_resizeLine" b:select="" -moz-opacity:0.6;
filter:alpha(opacity=60); border:4px solid #666;" />
    </s:event>
    <s:behavior b:name="remember_panel_cols">
        <s:event b:on="construct">
            <s:variable b:name="panelset_cols" b:select="cookie(concat(idpage(),
'panelset_cols'))" b:scope="local" />
            <s:task b:test="count($panelset_cols) eq 1" b:action="setpanelset"
b:cols="{$panelset_cols}" b:target.." />
        </s:event>
        <s:event b:on="resize">
            <s:task b:action="setcookie" b:name="{concat(idpage(), 'panelset_cols')}"
b:value="{$@:cols}" b:days="365" />
        </s:event>
    </s:behavior>
    <s:behavior b:name="remember_panel_rows">
        <s:event b:on="construct">
            <s:variable b:name="panelset_rows" b:select="cookie(concat(idpage(),
'panelset_rows'))" b:scope="local" />
            <s:task b:test="count($panelset_rows) eq 1" b:action="setpanelset"
b:rows="{$panelset_rows}" b:target.." />
        </s:event>
        <s:event b:on="resize">
            <s:task b:action="setcookie" b:name="{concat(idpage(), 'panelset_rows')}"
b:value="{$@:rows}" b:days="365" />
        </s:event>
    </s:behavior>
    <b:panelset b:cols="25% *">
        <b:panel id="panel_left" b:resize="e" b:behavior="remember_panel_cols"
b:minwidth="30px" b:resizeconstraint..">
            <div class="content_wrapper">
                <p>Add content....</p>
            </div>
        </b:panel>
        <b:panelset b:rows="40% *">
            <b:panel id="panel_top" b:resize="s" b:behavior="remember_panel_rows"
b:minheight="30px" b:resizeconstraint..">
                <div class="content_wrapper">
                    <p>Add content...</p>
                </div>
            </b:panel>
        <b:panel id="panel_bottom">

```

```

<div class="content_wrapper">
    <p>Add content...</p>
</div>
</b:panel>
</b:panelset>
</b:panelset>
</xmp>
```

## b:slider

A gauge/slider control extended with attributes that you can use to specify bounds, initial value, step size, snapping and orientation.

If you use the slider to specify a value of a field in a form, you must connect an input field with a slider control using the `b:connect` attribute whose value is an XPath expression to the slider, for example: `<input type="text" b:connect=".//b:slider" value="" />`

Set the `b:controlpath` on the `body` element to specify the controls you want to use.

### Attributes

#### b:orientation

Specify the orientation of the control, horizontal or vertical. Defaults to `horizontal`.

Values:

`horizontal`  
`vertical`

#### b:start

Set the minimum lower bounding value. The default is 1.

#### b:end

Sets the maximum upper bounding value. The default is 100.

#### b:step

Specify the step size which a value is incremented or decremented from a lower bound to an upper bound, and vice versa. The default is 1.

#### b:snap

Specify that the control has to snap to the positions allowed by the step size. The default is `false`.

Values:

`false`  
`true`

#### b:gfxset

Specify the `b:name` value of an `s:gfxset` that contains the images to use.

#### b:value

Specify an initial value of the slider on construction. The value is updated when you slide the thumb; you can read the current value of the slider within a `slide` event handler.

### Child Tags

#### b:slider-option

### Events

The `slide` event occurs when the user moves the thumb of a slider control. In the following example, when you move the slider the `slide` event handler in the behavior is triggered which reads out the current value of the `b:value` attribute and inserts it in the text node of the `span` id="readVariableValue" element: 50

## Implementation

In the following example, two sliders are defined: a horizontal slider whose value can be an even number between 0 and 100, and a slider that slides vertically from start value 50 to end value 100, and increases or decreases in multiples of 5. The `input` elements connect to the slider controls to display the values. The slider uses a behavior in which a `slide` event handler is defined: when you slide the control, the `b:value` attribute of the

read and inserted in the text node of the `span id="readVariableValue"` element:

```
<s:behavior b:name="slide">
  <s:event b:on="slide">
    <s:task b:action="set" b:target="id('currentvalue')/text()" b:value="@b:value" />
  </s:event>
</s:behavior>
<form>
  <span id="currentvalue">50</span>
  <b:slider b:behavior="slide" id="sliderHorizontal" b:value="50" b:orientation="horizontal" b:start="0" b:end="100" b:step="2" />
  <br />
  <input type="text" b:connect="id('sliderHorizontal')" value="" />
  <br />
  <br />
  <b:slider b:behavior="slide" id="sliderVertical" b:value="75" b:orientation="vertical" b:start="50" b:end="100" b:step="5" />
  <br />
  <input type="text" b:connect="id('sliderVertical')" value="" />
</form>
```

## b:slider-option

Defines an option for a slider control. You can provide the value inside the tag, or by specifying the `b:value` attribute.

### Attributes

#### `b:value`

A string containing a stated value.

### Parent Tags [Required]

#### [Required] `b:slider`

## Implementation

```
<b:slider id="slider" b:gfxset="slider-ver" b:snap="true">
  <b:slider-option>Warning</b:slider-option>
  <b:slider-option>Error</b:slider-option>
  <b:slider-option>Critical</b:slider-option>
</b:slider>
<input type="text" b:connect="id('slider')" />
```

## b:treelist

With the `treelist` control you can create trees with multiple columns. It is comparable to the newsgroup or threaded mail views in Outlook Express and Mozilla Thunderbird. The `b:treelist` tag is the root for such a control. `b:treelist` can also be used for simple, single-column trees.

If you want to use `treelist` with styling applied, you must add an `s:include` to your startup file and specify the `b:url` attribute to your control file, for example, `/controls/backbase/b-treelist/b-treelist.xml` directory).

### Child Tags

#### `b:treelistrow`

## Implementation

In the following example, the tree control has two levels. The `b:treelist` tag wraps around the elements that will show up in the tree. Expandable tree elements are created when:

1. A `b:treelistrow` tag contains more than one `b:treelistcell` child elements.
2. You place a `b:treelistchildren` node-set after a `b:treelistcell` element.

```
<b:treelist style="width:300px">
  <b:treelistrow>
```

```

<b:treelistcell>Entry 1</b:treelistcell>
<b:treelistcell>Description</b:treelistcell>
<b:treelistchildren>
  <b:treelistrow>
    <b:treelistcell>Sub-entry</b:treelistcell>
    <b:treelistcell>Description</b:treelistcell>
  </b:treelistrow>
</b:treelistchildren>
<b:treelistrow>
  <b:treelistcell>Entry 2</b:treelistcell>
  <b:treelistcell>Description</b:treelistcell>
  <b:treelistchildren>
    <b:treelistrow>
      <b:treelistcell>Sub-entry</b:treelistcell>
      <b:treelistcell>Description</b:treelistcell>
    </b:treelistrow>
  </b:treelistchildren>
</b:treelistrow>
<b:treelistrow>
  <b:treelistcell>Entry 3</b:treelistcell>
  <b:treelistcell>Description</b:treelistcell>
  <b:treelistchildren>
    <b:treelistrow>
      <b:treelistcell>Sub-entry</b:treelistcell>
      <b:treelistcell>Description</b:treelistcell>
    </b:treelistrow>
  </b:treelistchildren>
</b:treelistrow>
</b:treelist>

```

## b:treelistcell

The **treelistcell** represents one cell in the treelist. It is comparable to the HTML's `td` and within the `b:treelistcell` you can have any HTML or BXML tag.

Child Tags

---

`b:treelistrow`

Parent Tags [Required]

---

`b:treelistrow`

## b:treelistchildren

The `b:treelistchildren` tag is used to create the nested treelist items.

Child Tags

---

`b:treelistrow`

Parent Tags [Required]

---

`b:treelistrow`

## b:treelistrow

The **treelistrow** defines a row in the treelist. It is displayed like an HTML `tr` tag. Within the `b:treelistrow`, you can have a number of `b:treelistcell` tags each defining a column in the treelist row. The tree controls will appear in the first cell. Also, you can use a `b:treelistchildren` tag as a container for nested entries.

For a `treelistrow`, you can also use `b:label` as a shorthand alternative for the required `treelistcell`.

Attributes

---

`b:label`

Specify a String value that displays text in a set location for a control.

---

`b:gfxset`

Specify the `b:name` value of an `s:gfxset` that contains the images to use.

Child Tags**b:treelistcell , b:treelistchildren**Parent Tags [Required]**b:treelist , b:treelistchildren**

## b:xhtml

Marks the start of the ***HTML space***. All tags inside are ignored by the BPC parser, except for **b:bxm1** tags that mark the ***BXML space*** within the **b:xhtml** tags. It is recommended to place non-dynamic content, such as text documents, inside the HTML space for better efficiency and to maximize memory utilization.

Attributes**b:focustype**

Make the cells of a table in the HTML space focusable. Each cell of the table is then a focusitem that can get focus using the keyboard arrow keys. The **bpc\_focusCurrentElement** variable contains the focusitem in the HTML space that currently has focus (is null when no element has focus), and the **bpc\_focusLastElement** variable contains the focus item that previously has focus (is set when the **blur** event is fired).

Values:  
htmltable

## Implementation

The following example shows a table defined within the HTML space in which the focus model has been implemented. Note also the **focus** event is triggered when the focus item (a cell in the table) is selected:

```
<s:behavior b:name="x">
    <s:event b:on="command" b:action="alert" b:value="command" />
    <s:event b:on="focus" b:action="alert" b:value="focus" />
</s:behavior>
<b:xhtml b:focusitem="true" b:focusindicator="true" b:focustype="htmltable" b:behavior="x">
    <table border="1">
        <tbody>
            <tr>
                <td>A1</td>
                <td>A2</td>
                <td>A3</td>
                <td>A4</td>
            </tr>
            <tr>
                <td>B1</td>
                <td>B2</td>
                <td>B3</td>
                <td>B4</td>
            </tr>
            <tr>
                <td>C1</td>
                <td>C2</td>
                <td>C3</td>
                <td>C4</td>
            </tr>
        </tbody>
    </table>
</b:xhtml>
```

## BXML S Tags

### s:apply-templates

The `s:apply-templates` template mechanism tag applies the specified template rules to the current element or to its child nodes.

#### Attributes

##### b:name

Specify the name of a template.

##### b:select

Specify the element that is selected using an XPath statement.

#### Parent Tags [Required]

`s:element` , `s:for-each` , `s:if` , `s:otherwise` , `s:template` , `s:value-of` , `s:variable` , `s:when`

### s:attribute

The `s:attribute` is a template mechanism tag that adds an attribute to the specified element in the output document.

#### Attributes

##### [Required] b:name

Specify a logical unique name with which to identify an element.

##### b:select

Specify the element that is selected using an XPath statement.

#### Child Tags

`s:apply-templates` , `s:attribute` , `s:call-template` , `s:choose` , `s:copy-of` , `s:element` , `s:for-each` , `s:fxstyle` , `s:history` , `s:if` , `s:lock` , `s:parallel` , `s:render` , `s:script` , `s:sequential` , `s:setatt` , `s:setstyle` , `s:task` , `s:value-of` , `s:variable` , `s:with`

#### Parent Tags [Required]

`s:bookmark` , `s:element` , `s:event` , `s:for-each` , `s:history` , `s:httpcode` , `s:if` , `s:if` , `s:keys` , `s:lock` , `s:otherwise` , `s:otherwise` , `s:parallel` , `s:sequential` , `s:tasklist` , `s:template` , `s:value-of` , `s:variable` , `s:when` , `s:when` , `s:with`

### s:attribute

The `s:attribute` is an htmlstructure tag that adds an attribute to the specified element in the structure. When the value of this attribute is taken from the custom tag, it is automatically synchronized.

#### Parent Tags [Required]

#### Attributes

##### [Required] b:name

Specify a logical unique name with which to identify an element.

##### [Required] b:value

A string containing a stated value.

### s:behavior

A **behavior** is a generic construct used to encapsulate functionality for reuse, and to separate the structure of the document from the behavior. You can use behaviors to do the following:

- Define the visual representation of an element when in one of its base states, **selected** or **deselected**, using the `s:state` tags
- Define the instructions (tasks and commands) that are executed when an event

takes place (using the `s:event` tag)

For more information on events, see [Defining Event Handlers \[Backbase Manual\]](#)

- Define the keys that active within an element `s:whenactive`
- Set the initial attributes for an element using `s:initatt`

To use a behavior on an XHTML or a BXML element, you need to add the `b:behavior` attribute with the value of the behavior's name to the element.

#### Attributes

##### [Required] `b:name`

Specify a logical unique name with which to identify an element.

##### `b:behavior`

Specifies the name of a behavior from which it inherits all event handlers. If the child behavior has event handlers for the same event as the parent, the event handlers on the child behavior override those of the parent, unless you use the the `s:super` tag.

#### Child Tags

`s:event` , `s:initatt` , `s:state` , `s:whenactive`

### Implementation

This example shows how to change CSS classes for a given element. The styling of the elements is dependent on whether they are selected or deselected. Clicking on an element will flip its state (`selected` or `deselected`). For each one of the two states, two CSS classes are made that change the background color.

```
<s:behavior b:name="simpleBehavior">
    <s:state b:on="deselect" b:normal="myclass deselect-normal" b:hover="myclass
deselect-hover" />
    <s:state b:on="select" b:normal="myclass select-normal" b:hover="myclass select-hover" />
    <s:event b:on="command">
        <s:task b:action="select-deselect" />
    </s:event>
</s:behavior>
<style type="text/css">      .myclass { border: 1px solid #888; width:300px; }
.myclass.deselect-normal { background: #FFF; }      .myclass.select-normal { background:
#DDD; }
.myclass.deselect-hover { background: #BBB; }      .myclass.select-hover { background: #666; }</style>
<p b:behavior="simpleBehavior" b:state="selected">Behavior applied to this tag</p>
<p b:behavior="simpleBehavior" b:state="selected">Behavior applied to this tag</p>
<p b:behavior="simpleBehavior">Behavior applied to this tag</p>
<p b:behavior="simpleBehavior">Behavior applied to this tag</p>
```

### s:bookmark

The bookmark tag is executed on startup of the application, after the execution of `construct` events and `s:execute` tags. Inside the bookmark tag you can place code to make sure the bookmark is handled correctly.

#### Child Tags

`s:choose` , `s:fxstyle` , `s:history` , `s:if` , `s:lock` , `s:parallel` , `s:render` , `s:script` , `s:sequential` , `s:setatt` , `s:setstyle` , `s:task` , `s:variable` , `s:with`

### Implementation

In the following example, you can bookmark each tab in the tabbox. When you click the tabs, note that the URL changes, for example `...#SevenSamurai[2]`, allowing you to bookmark it. Each tab uses a behavior called history. When a tab is selected, the `s:history` tag creates a new history item for the tab, using the `id` attribute to bookmark the tab. The `b:name="browser"` attribute enables the user to navigate back and forward using the browser history buttons.

The `construct` event is triggered when the application is (re)loaded. When you have

bookmarked for example the tab called Seven Sumurai, the tab is selected using the ID the bookmark points to. The value of the bookmark is accessed using the `$bpc_bookmark` XPath variable.

The files used in this example are available in your Backbase installation `explorer/data` directory

```
<s:bookmark>
  <s:task b:action="select" b:target="id($bpc_bookmark)" />
</s:bookmark>
<s:behavior b:name="history" b:behavior="b-tab">
  <s:event b:on="select">
    <s:super />
    <s:history b:name="browser" b:bookmark="@{id}">
      <s:task b:action="select" />
    </s:history>
  </s:event>
</s:behavior>
<b:tabbox style="height: 100%">
  <b:tab b:label="2001" id="2001" b:url="data/2001_short.xml" b:behavior="history" />
  <b:tab b:label="Seven Samurai" id="SevenSamurai" b:url="data/seven_short.xml" b:behavior="history" />
  <b:tab b:label="Life is Beautiful" id="LifeisBeautiful" b:state="selected" b:url="data/bella_short.xml" b:behavior="history" />
  <b:tab b:label="The Godfather" id="Godfather" b:url="data/godfather_short.xml" b:behavior="history" />
</b:tabbox>
```

Note: The `b:behavior="b-tab"` attribute and the `s:super` tag in this example are there to make sure the tabbed navigation control works properly, they are not related to history or bookmarks. If you want to have back-button history without bookmarks, simply use the `s:history` without `b:bookmark` attribute.

## **s:call-template**

The `s:call-template` instruction invokes a template by name; it has a required `name` attribute that identifies the template to be invoked.

### Attributes

[Required] `b:name`

Specify the name of a template.

### Parent Tags [Required]

`s:element` , `s:for-each` , `s:if` , `s:otherwise` , `s:template` , `s:value-of` , `s:variable` , `s:when`

## **s:choose**

Lists a number of choices whose contents will be executed if the tests return true. If none of the `s:when` tests succeed, the contents of the `s:otherwise` element will be executed.

### Child Tags

`s:otherwise` , [Required] `s:when`

### Parent Tags [Required]

`s:bookmark` , `s:element` , `s:event` , `s:for-each` , `s:history` , `s:httpcode` , `s:if` , `s:if` , `s:keys` , `s:lock` , `s:otherwise` , `s:otherwise` , `s:parallel` , `s:sequential` , `s:tasklist` , `s:template` , `s:value-of` , `s:variable` , `s:when` , `s:when` , `s:with`

## **s:controlpath**

You can use the `s:controlpath` to specify individual control files (containing the implementation of your controls) that are loaded for the controls that you have used in your application. It allows you to, for example, mix the Backbase skins you use for individual

controls, and to locate your custom controls in separate directories.

The `s:controlpath` tag overrides the `b:controlpath` attribute: you can set a control path using the `b:controlpath` attribute on the `body` element. This sets a path to load controls automatically (the default path is the `controls/dynamic` folder).

### Attributes

#### [Required] `b:match`

Specify the match pattern for the controlpath.

#### [Required] `b:value`

A string containing a stated value.

### Implementation

In the following example, the `b:match="b:bo"` matches the `b:box` control but not `b:button`, therefore the box control uses the Basic skin and the Button the backbase skin (which is the default control in the `/dynamic` folder, assuming no `b:controlpath` has been set). The `b:match="xxx:"` shows you how you can include all your custom controls defined, for example, in the `xxx` namespace.

```
<s:controlpath b:match="b:bo" b:value=".../controls/basic/" />
<s:controlpath b:match="xxx:" b:value=".../controls/mystuff/" />
<s:controlpath b:match="b:my" b:value=".../controls/mystuff/" />
<b:box>
  <b:button>Click me!</b:button>
</b:box>
```

## S:copy-of

The `s:copy-of` tag is used to copy a set of nodes to the result tree.

### Parent Tags [Required]

#### [Required] `s:render`

### Attributes

#### [Required] `b:select`

Specify the element that is selected using an XPath statement.

## S:copy-of

The `s:copy-of` template mechanism tag creates a copy of the current node (with child nodes and attributes).

### Attributes

#### [Required] `b:select`

Specify the element that is selected using an XPath statement.

### Parent Tags [Required]

`s:element` , `s:for-each` , `s:if` , `s:otherwise` , `s:template` , `s:value-of` , `s:variable` , `s:when`

## S:default

The `s:default` tag is used to set a default `class` or a default `behavior` to a tag. This can be used to give tags a default representation, the Backbase skin files use this tag to style the common components of a web application.

### Attributes

#### [Required] `b:tag`

A tag name, including its namespace prefix.

#### Values:

`b:slider`  
`b:treelist`

#### [Required] `b:attribute`

Specify a string with the name of an element's attribute. The `s:default` tag accepts the values `b:behavior` or `class`.

**[Required] `b:value`**

A string containing a stated value.

**Child Tags**

Child tags are not allowed in this tag.

## Implementation

The following code globally applies a default representation to all `div` elements. All occurrences of `<div>...</div>` would have the same result as if you would extend it with the `class` attribute as in `<div class="my-div-class">...</div>`.

```
<style type="text/css">      .my-div-class {      background-color: red;      color: white;  }</style>
<s:default b:tag="div" b:attribute="class" b:value="my-div-class" />
<div>The text will be white on a red background</div>
```

## **s:element**

The `s:element` template mechanism tag creates an element node in the output document.

**Attributes**

**[Required] `b:name`**

Sets the name of the element.

**Child Tags**

`s:apply-templates` , `s:attribute` , `s:call-template` , `s:choose` , `s:copy-of` , `s:element` , `s:for-each` , `s:if` , `s:value-of` , `s:variable`

**Parent Tags [Required]**

`s:element` , `s:for-each` , `s:if` , `s:otherwise` , `s:template` , `s:value-of` , `s:variable` , `s:when`

## **s:event**

The `s:event` tag is used to define an event handler. The event handler defines what should happen (the command, or actions that take place) when a certain event is fired. An event can be triggered by a user, such as `click` or `mouseenter`, or a system event such as `construct` (for more information on available events, see the Events and Commands sections). You can define events inside any user interface element, or define it within a behavior. The `s:event` element is triggered by the event specified by the `b:on` attribute and can have any executable tag, such as `s:task` or `s:fxstyle`, as its child tags (see below). For detailed information on defining event handler, see the `manual.pdf` in the Backbase installation `/docs` directory.

**Attributes**

**[Required] `b:on`**

Specify the name of the event on which the actions are triggered.

**`b:action`**

Specify the command to execute when the element's command event fires, for example when it is clicked. Each command has specific attributes. See the command section for an overview of all the commands.

**`b:test`**

Specify a condition, expressed in XPath, that returns either true or false. For example, `b:test="@border = '0'"` checks whether the border attribute of the context node is set to 0; if it isn't, the code is not executed.

**`b:async`**

Specify that the commands within a task will run asynchronously. The default

is `false`. If set to `true`, a tag extended with this attribute tells the BPC engine that the response of the containing action can be ignored for the rest of the process.

Values:  
`false`  
`true`

#### b:bubble

Registers an event triggered on an element and passes it on to the current node's ancestors. The event bubbles up the object hierarchy, unless it encounters a node on which the bubble attribute is set to `false`. By default, the value is set to `true`. Event bubbling only applies to user input events.

---

#### Child Tags

`s:choose` , `s:fxstyle` , `s:history` , `s:if` , `s:lock` , `s:parallel` , `s:render` , `s:script` , `s:sequential` , `s:setatt` , `s:setstyle` , `s:super` , `s:task` , `s:variable` , `s:with`

---

#### Parent Tags

`s:behavior`

## Implementation

The following example defines a simple event handler defined inline in a link element. Clicking the link triggers the `command` event. The event handler defines a task: the `b:action="alert"` specifies the command to execute when the element's command event fires.

```
<a>Click me
  <s:event b:on="command">
    <s:task b:action="alert" b:value="Hello" />
  </s:event>
</a>
```

Note that the `command` event is triggered by default. You can therefore write the above example in shorthand as follows:

```
<a b:action="alert" b:value="Hello">Click me</a>
```

## S:execute

The `s:execute` tag executes its children immediately after being parsed, after `construct` events have been executed. Therefore, if you define `s:execute` at the root of a response file loaded by a *form submission* or a `load` command, the data is NOT inserted at the specified destination, but is executed immediately.

---

#### Attributes

##### b:context

Specify the context in which the task is executed. The attribute associates the task (`s:execute` or `render` command) with a BXML element. The value is an XPath expression, for example `b:context="//*"` defines the root element. By default, the context is the parent of the task element.

---

#### Child Tags

`s:choose` , `s:fxstyle` , `s:history` , `s:if` , `s:lock` , `s:parallel` , `s:render` , `s:script` , `s:sequential` , `s:setatt` , `s:setstyle` , `s:task` , `s:variable` , `s:with`

## s:for-each

The `for-each` tag can be used to select every XML element of a specified node-set.

---

#### Attributes

##### [Required] b:select

Specify the element that is selected using an XPath statement.

---

#### Child Tags

`s:apply-templates` , `s:attribute` , `s:call-template` , `s:choose` , `s:copy-of` ,

s:element , s:for-each , s:if , s:value-of , s:variable

#### Parent Tags [Required]

s:element , s:for-each , s:if , s:otherwise , s:template , s:value-of , s:variable , s:when

## S:fxstyle

Defines an **effect** for an element. You typically define effects in an event handler defined within a **behavior** that is then used by the element. You can use attributes in **s:fxstyle** that represent CSS properties by prefixing the name with **b:**. The current value of that CSS property is then animated to the given value.

If you want the new value to be relative to the current value, use the tilde '~' sign. For example, to increase the CSS left property of an element by 100 pixels, use a value of **b:left="~100px"**, and to decrease the CSS left property by 100 pixels, use a value of **b:left="~-100px"**.

By default, if you have defined several **s:fxstyle** tags within an event handler, these are executed sequentially (one-by-one). To execute them asynchronously, use the **s:parallel** tag.

You cannot animate CSS properties such as font-family, text-decoration, and background-image. Any word-based values, such as **Arial**, **Orange**, or **Inherit**, are ignored, except **auto** when used for width or height values.

#### Attributes

##### b:motion

Define how an element changes position or location; with **linear**, the change is equally-spaced steps, with **log** the change is logarithmic, and with **exp** the change is exponential. By default, the value is set to **linear**. You can also define your own JavaScript function to define the motion.

##### Values:

```
<name_of_Javascript_function>
exp
linear
log
```

##### b:test

Specify a condition, expressed in XPath, that returns either true or false. For example, **b:test="@border = '0'"** checks whether the border attribute of the context node is set to 0; if it isn't, the code is not executed.

##### b:cancel

Set the value to **false** to prevent the user from being able to stop an **s:fxstyle** effect.

##### b:time

Specify the duration, in milliseconds, of an **s:fxstyle** effect.

##### b:background-color

Sets the background color of an element, either a **color** value or the keyword **transparent** to make the underlying colors shine through.

##### Values:

```
<color>
inherit
transparent
```

##### b:border-color

Specify the color of a box's border. Border values **{1,4}** refer respectively to top, right, bottom, and left border. For example, **{border-color: red green blue}** results in red top border, green left and right border, and, blue bottom border. **{border-color: red green blue yellow}** results in red top and bottom border, and green left and right border.

##### Values:

```
<color> | transparent | {1,4} | inherit
```

**b:border-top-color**

Specify the color of the top edge of a box's border.

Values:

<color>  
inherit  
transparent

**b:border-right-color**

Specify the color of the right edge of a box's border.

Values:

<color>  
inherit  
transparent

**b:border-bottom-color**

Specify the color of the bottom edge of a box's border.

Values:

<color>  
inherit  
transparent

**b:border-left-color**

Specify the color of the left edge of a box's border.

Values:

<color>  
inherit  
transparent

**b:border-top-width**

Specify the width of the top edge of a box's border.

Values:

<length>  
inherit  
medium  
thick  
thin

**b:border-right-width**

Specify the width of the right edge of a box's border.

Values:

<length>  
inherit  
medium  
thick  
thin

**b:border-bottom-width**

Specify the width of the bottom edge of a box's border.

Values:

<length>  
inherit  
medium  
thick  
thin

**b:border-left-width**

Specify the width of the left edge of a box's border.

**b:border-width**

A shorthand property used for setting the width of the border area; border-top-width, border-right-width, border-bottom-width, and border-left-width. If there is only one value, it applies to all sides. If there are two values, the top and bottom borders are set to the first value and the right and left are set to the second. If there are three values, the top is set to the first value, the left and right are set to the second, and the bottom is set to the third. If there are four values, they apply to the top, right, bottom, and left, respectively. (If you specify a length, set an explicit value (must not be negative.)

Values:

<length>  
inherit  
medium  
thick  
thin

**b:bottom**

A percentage or pixel value that sets the bottom position from the bottom edge of the containing element. Negative values are allowed.

#### b:color

Describe the foreground color of an element's text content.

Values:  
 <color>  
 inherit

#### b:font-size

Set the size of the font for an element.

Values:  
 <length>  
 <percentage>  
 inherit  
 large  
 larger  
 medium  
 small  
 smaller  
 x-large  
 x-small  
 xx-large  
 xx-small

#### b:height

Specify a percentage or pixel value for the height of an element.

#### b:left

Specify a percentage or pixel value that sets the left position from the left edge of the containing element. Negative values are allowed.

#### b:letter-spacing

Define the amount of whitespace to be inserted between the character boxes of text.

Values:  
 <length>  
 inherit  
 normal

#### b:line-height

Influence the layout of line boxes.

Values:  
 <length>  
 <number>  
 <percentage>  
 inherit  
 normal

#### b:margin

A shorthand property that sets the margin for all four sides. {1,4} refers to respectively margin-top, margin-right, margin-bottom, and margin-left. (The other margin properties only set their respective side.)

Values:  
 [<length> | <percentage> | auto ] {1,4}  
 inherit

#### b:margin-top

Set the width of a top margin for an element.

Values:  
 <length>  
 <percentage>  
 auto  
 inherit

#### b:margin-right

Set the width of a right margin for an element.

Values:  
 <length>  
 <percentage>  
 auto  
 inherit

#### b:margin-bottom

Set the width of a bottom margin for an element.

Values:

```
<length>
<percentage>
auto
inherit
b:margin-left
```

Set the width of a left margin for an element.

Values:  
 <length>  
 <percentage>  
 auto  
 inherit

**b:padding**

A shorthand property for setting all padding properties. {1,4} refers respectively to padding-top, padding-right, padding-bottom, and padding-left. For example, {padding: 0.5cm 2.5cm} sets a top and bottom padding of 0.5cm, and a left and right padding of 2.5cm.

Values:  
 [<length> | <percentage>] {1,4}  
 inherit

**b:padding-top**

Set the width of the top padding for an element.

Values:  
 <length>  
 <percentage>  
 inherit

**b:padding-right**

Set the width of the right padding for an element.

Values:  
 <length>  
 <percentage>  
 inherit

**b:padding-bottom**

Set the width of the bottom padding for an element.

Values:  
 <length>  
 <percentage>  
 inherit

**b:padding-left**

Set the width of the left padding for an element.

Values:  
 <length>  
 <percentage>  
 inherit

**b:right**

Specify a percentage or pixel value that sets the right position from the right edge of the containing element. Negative values are allowed.

**b:scroll-left**

Scrolls to the left.

**b:scroll-top**

Scrolls to the top.

**b:text-indent**

Define the indentation of the first line of text in a block-level element.

Values:  
 <length>  
 <percentage>  
 inherit

**b:top**

Specify a percentage or pixel value that sets the top position from the top edge of the containing element. Negative values are allowed.

**b:width**

Specify a percentage or pixel value specifying the width of an element.

**b:word-spacing**

Declare how whitespace inside the element is handled.

Values:

```
inherit
normal
nowrap
pre
pre-line
pre-wrap
```

**Child Tags**

Child tags are not allowed in this tag.

**Parent Tags [Required]**

s:bookmark , s:event , s:history , s:httpcode , s:if , s:keys , s:lock , s:otherwise  
, s:parallel , s:sequential , s:tasklist , s:when , s:with

**Implementation**

The following is a simple example of an effect that is triggered when the mouse enters and leaves the `div` element:

```
<div>
    <s:event b:on="mouseenter">
        <s:fxstyle b:padding-top="10px" b:background-color="#FF0000" b:padding-left="10px"
b:color="#FFFFFF" />
    </s:event>
    <s:event b:on="mouseleave">
        <s:fxstyle b:padding-top="2px" b:background-color="#FFFFFF" b:padding-left="2px"
b:color="#000000" />
    </s:event>
    CONTENT
</div>
```

In the following example, a custom JavaScript function is used to determine the motion type. You call the JavaScript function by its name in the `b:motion` attribute. The custom JavaScript function has the following parameters:

- `startValue` - an integer representing the initial value of the CSS property that will be manipulated.
- `endValue` - integer representing the final value of the CSS property that will be manipulated.
- `endTime` - integer representing the time that the fxstyle will finish in milliseconds (this is a very large value).
- `startTime` - integer representing the time that the fxstyle will start in milliseconds (this is a very large value).
- `currentTime` - integer representing the current time in milliseconds (this is a very large value).

The function returns an integer value that is used to assign a new value to the CSS property for the current step of the transition. The function is called repeatedly until the allotted `b:time` for the transition has passed.

```
<script type="text/javascript">
    function motion_jerky(iValueStart, iValueEnd, iTimeEnd, iTimeStart, iTimeCurrent) {
        //create 8 equally sized steps
        var iStepSize = (iValueEnd - iValueStart) / 8;
        //calculate the exact position
        var newExactPos = iValueStart + (iValueEnd - iValueStart) * (iTimeCurrent -
iTimeStart) / (iTimeEnd - iTimeStart);
        //round the exact position into steps
        return parseInt(newExactPos / iStepSize) * iStepSize;
    }</script>
```

```
<s:behavior b:name="move_test">
    <s:event b:on="click">
        <s:lock>
```

```

        <s:fxstyle b:motion="motion_jerky" b:left="~-400px" b:time="4000" />
        <s:fxstyle b:motion="linear" b:left="~400px" b:time="4000" />
    </s:lock>
</s:event>
</s:behavior>
<div b:behavior="move_test" style="position:absolute; border:1px solid red; top:400px;
left:500px"> Click me!</div>

```

## s:gfx

The **s:gfx** tag is the child tag of an **s:gfxset** tag, which specifies the graphic to use depending on the state the element is in the interface.

### Attributes

#### [Required] b:type

In the context of the **s:gfx** element, the **open** (plus button image), the **closed** (minus button image), the **leaf** (image displayed when a **b:treelistcell** has no further children) and the **undefined** (image displayed when the tree node has not yet been clicked upon). The **grippy** (grippy button) value is used for the slider control.

#### Values:

close  
grippy  
leaf  
open  
undefined

#### [Required] b:normal

Specify the style class(es) to use when the element is in its base state.

### Child Tags

Child tags are not allowed in this tag.

### Parent Tags [Required]

**s:gfxset**

## s:gfxset

The **s:gfxset** tag allows you to supply your own images for a slider or treelist control. It does not need to supply all the properties for a control, since it will inherit and override the default properties.

### Attributes

#### [Required] b:name

Specify a logical unique name with which to identify an element.

#### [Required] b:tag

A tag name, including its namespace prefix.

#### Values:

b:slider  
b:treelist

#### [Required] b:directory

Define a custom directory path for the **s:gfxset** tag. It contains a string that is prefixed to file names.

### Child Tags

[Required] **s:gfx**

## Implementation

In the following example, the **b:treelistrow** element is extended with the **b:gfxset** attribute will show two different custom graphics instead of the default graphics: one graphic named *g\_tree\_closed.gif* for when it is collapsed, another named *g\_tree\_open.gif* for when it is expanded. The value of the **b:gfxset** attribute determines which graphic set instructions to use.

```

<s:gfxset b:directory=". " b:tag="b:treelist" b:name="green">
  <s:gfx b:type="open" b:normal="up.gif" />
  <s:gfx b:type="closed" b:normal="down.gif" />
</s:gfxset>
<b:treelist style="width:300px">
  <b:treelistrow>
    <b:treelistcell>Entry 1</b:treelistcell>
    <b:treelistcell>Description</b:treelistcell>
    <b:treelistchildren>
      <b:treelistrow b:gfxset="green">
        <b:treelistcell>Entry 2</b:treelistcell>
        <b:treelistcell>Description</b:treelistcell>
        <b:treelistchildren>
          <b:treelistrow>
            <b:treelistcell>Entry 3</b:treelistcell>
            <b:treelistcell>Description</b:treelistcell>
          </b:treelistrow>
        </b:treelistchildren>
      </b:treelistrow>
    </b:treelistchildren>
  </b:treelistrow>
</b:treelist>
</b:treelist>

```

## s:history

The `s:history` tag allows you to program the browser backwards and forwards navigation, and to create your own history stacks. The browser back/forward mechanism is handled by specifying the (required) attribute `b:name="browser"`. You can create your own mechanism by specifying a different name for the attribute. The `s:history` tag is useful for elements that are part of a singular system, for example a deck, where only one of the children can be seen at any one time.

### Child Tags

`s:choose` , `s:fxstyle` , `s:history` , `s:if` , `s:lock` , `s:parallel` , `s:render` , `s:script` , `s:sequential` , `s:setatt` , `s:setstyle` , `s:task` , `s:variable` , `s:with`

### Parent Tags [Required]

`s:bookmark` , `s:event` , `s:history` , `s:httpcode` , `s:if` , `s:keys` , `s:lock` , `s:otherwise` , `s:parallel` , `s:sequential` , `s:tasklist` , `s:when` , `s:with`

### Attributes

[Required] `b:name`

Specify a logical unique name with which to identify an element.

## s:htmlstructure

The `s:htmlstructure` tag is for developing a user-defined structure. Defining an HTML structure is a method for abstracting HTML elements, such as UI controls, into a single custom tag to build common structures promoting reuse (HTML structures are used in making the out-of-the-box Backbase controls).

The `s:htmlstructure` tag is the main tag that is used to make a structure definition. The contents of the structure are then inserted in your document using the custom tag, in other words the `b:name` attribute of the `s:htmlstructure`. An `s:htmlstructure` definition must **only** consist of HTML elements, and the Backbase tags: `s:value-of`, `s:attribute`, and `s:innercontent` /. The `s:innercontent` / child tag is a placeholder to determine the location where the contents of the invoking custom tag are placed. It must always be the only child of its parent.

The `s:htmlstructure` tag requires a namespace identifier, for example `<s:htmlstructure b:name="c:mytag"/>`. You must then set a namespace declaration on a container or parent element in which the structure is used, for example `xm-`

```
lns:c="http://www.backbase.com/c".
```

Note: The content of the structure definition is in the HTML space, and not in BXML space. You cannot insert BXML tags, or target elements in the BXML space. The behaviors in the HTML space have restricted functionality.

#### Attributes

##### [Required] b:name

Specify a unique name with which to identify a custom tag, defined using the s:htmlstructure tag.

##### b:behavior

Attaches a *behavior*, defined elsewhere within s:behavior tags, to the element. The value is the same as the b:name value of the behavior.

#### Child Tags

You can define any HTML elements as children of this tag. You can also use the following tag-specific child tags:

s:attribute , s:innercontent , s:value-of

#### Implementation

In the following example, the s:htmlstructure tag is used to create a custom tag - in this case some div tags with defines styles (the contents of the tags appear as boxes with a thick grey border).

```
<s:htmlstructure b:name="c:mytag" b:behavior="x">
    <div style="display: inline; background-color: #CCC; padding: 10px 5px; margin: 5px;">
        <div style="display: inline; background-color: #EEE; padding: 5px">
            <s:innercontent />
        </div>
    </div>
</s:htmlstructure>
<s:behavior b:name="x">
    <s:event b:on="command" b:action="alert" b:value="haha" />
</s:behavior>
```

Use the custom tag as follows (at runtime, the content of the c:mytag tag is placed in the s:innercontent /> tag of the HTML structure):

```
<div style="margin: 10px">
    <c:mytag>Some text</c:mytag>
    <c:mytag>Some
        <b>html</b>
    </c:mytag>
    <c:mytag>Some BXML
        <s:tooltip>BXML = Backbase XML</s:tooltip>
    </c:mytag>
</div>
```

#### s:httppcode

The *httppcode* tag is used to catch HTTP error codes.

#### Attributes

##### [Required] b:type

In the context of the s:httppcode tag, it is used to indicate the number of the HTTP error code which should be caught.

#### Child Tags

s:choose , s:fxstyle , s:history , s:if , s:lock , s:parallel , s:render , s:script , s:sequential , s:setatt , s:setstyle , s:task , s:variable , s:with

#### Implementation

The following code demonstrates how to catch errors thrown while retrieving files from the server:

```
<s:httppcode b:type="404">
```

```

<s:task b:action="alert" b:value="The file could not be found!" />
</s:httphandler>
<a b:action="load" b:target="#loadarea" b:url="doesnotexist"> Click here to load a
file that does not exist...</a>
<div id="loadarea"></div>

```

## s:httphandler

The `s:httphandler` tag allows you to create custom HTTP header information. You must define the headers in a `load` command.

### Attributes

[Required] `b:name`

Specify a logical unique name with which to identify an element.

[Required] `b:value`

A string containing a stated value.

### Child Tags

`s:choose` , `s:fxstyle` , `s:history` , `s:if` , `s:lock` , `s:parallel` , `s:render` , `s:script` ,  
`s:sequential` , `s:setatt` , `s:setstyle` , `s:task` , `s:variable` , `s:with`

## Implementation

```

<s:task b:action="load" b:url="clean.xml" b:destination="id('x')">
  <s:httphandler b:name="one" b:value="two" />
  <s:httphandler b:name="three" b:value="four" />
</s:task>

```

## S:if

The tasks within the `s:if` tag are executed only if the `b:test` returns true.

The shorthand notation for defining commands (the `b:action` attribute) for `s:when`, `s:otherwise`, `s:if` is not supported. You must define the action to be taken within a `s:task` element. For example, the construction `s:if b:test="" b:action="commandName" /` is not supported; instead use the following:

```

<s:if b:test="">
  <s:task b:action="commandName" />
</s:if>

```

### Attributes

[Required] `b:test`

Specify a condition, expressed in XPath, that returns either true or false. For example, `b:test="@border = '0'"` checks whether the border attribute of the context node is set to 0; if it isn't, the code is not executed.

### Child Tags

`s:apply-templates` , `s:attribute` , `s:call-template` , `s:choose` , `s:copy-of` ,  
`s:element` , `s:for-each` , `s:fxstyle` , `s:history` , `s:if` , `s:lock` , `s:parallel` , `s:render` ,  
`s:script` , `s:sequential` , `s:setatt` , `s:setstyle` , `s:super` , `s:task` , `s:value-of` ,  
`s:variable` , `s:with`

### Parent Tags [Required]

`s:bookmark` , `s:element` , `s:event` , `s:for-each` , `s:history` , `s:httphandler` , `s:if` , `s:if` ,  
`s:keys` , `s:lock` , `s:otherwise` , `s:otherwise` , `s:parallel` , `s:sequential` , `s:tasklist` ,  
`s:template` , `s:value-of` , `s:variable` , `s:when` , `s:when` , `s:with`

## s:include

The `s:include` tag loads the XML file specified in the `b:url` attribute, and replaces itself with the parsed content of that file.

The `s:include` tag does not work across domains. You can only include files located on the same host as the your web application (otherwise you have to write your own serv-

er-side proxy).

#### Attributes

##### [Required] b:url

Specify the path of the file to load, relative to the startup file of your Backbase application.

#### Child Tags

Child tags are not allowed in this tag.

### Implementation

In the following example, the `s:include` tag is replaced with the contents of the `form.xml` file located in the `data` folder (the path to the file is relative to the current startup page):

```
<s:include b:url="data/form.xml" />
```

## s:initatt

Sets the initial attributes on an element when the page is rendered by the BPC. Note that attributes are not overwritten, only added when they do not exist.

### Implementation

```
<s:behavior b:name="initatt">
  <s:initatt b:resize="all" b:style="background: red" />
</s:behavior>
<b:box b:behavior="initatt">
  <div>When the page is rendered, the s:initatt sets initial attributes.</div>
</b:box>
```

#### Parent Tags

##### s:behavior

## s:innercontent

The `s:innercontent` tag used only inside an `s:htmlstructure` tag as a placeholder for the content of the custom tag with which a *structure* can be applied (see `s:htmlstructure`).

Note that the `s:innercontent` tag must be the **only** child of its parent element. When you want to place multiple children in the parent, you can surround the `s:innercontent` tag with a `div` or a `span` tag.

#### Parent Tags [Required]

##### s:htmlstructure

#### Child Tags

Child tags are not allowed in this tag.

## s:keys

Maps a command to a key, or keys. When the key(s) defined by the `b:keys` attribute is pressed, the command defined by the `b:action` is executed. The `s:whenactive` tag is a wrapper for the `s:keys` tag to provide the application context for when the keys are enabled; the keys are only enabled when the elements to which they are applied are active. You can define keys inline, or inside a behavior (but they must always be contained in a `s:whenactive`). For example, to define global keys, you can define them in a `s:behavior` that is used in the `body` element to be active throughout your application.

Input element have default focus keys assigned to them. To use a different keys for jumping to different elements, you need to overrule these active keys using, for ex-

ample, the following syntax; <s:keys b:keys="tab" b:bubble="false" />

### Attributes

#### [Required] b:keys

Bind a key, or keys, to a command. The value is either the character of the key (case insensitive), or decimal ASCII value of the key. Use a space-separated list to define more than one key, for example `b:keys="down ctrl+down shift+down"`

#### b:action

Specify the command to execute when the element's command event fires, for example when it is clicked. Each command has specific attributes. See the command section for an overview of all the commands.

#### b:test

Specify a condition, expressed in XPath, that returns either true or false. For example, `b:test="@border = '0'"` checks whether the border attribute of the context node is set to 0; if it isn't, the code is not executed.

#### b:bubble

Registers an event triggered on an element and passes it on to the current node's ancestors. The event bubbles up the object hierarchy, unless it encounters a node on which the bubble attribute is set to `false`. By default, the value is set to `true`. Event bubbling only applies to user input events.

### Parent Tags [Required]

#### [Required] s:whenactive

### Child Tags

s:choose , s:fxstyle , s:history , s:if , s:lock , s:parallel , s:render , s:script , s:sequential , s:setatt , s:setstyle , s:task , s:variable , s:with

## Implementation

The following example describes how you can overrule the default keys for focus elements. The text areas defined use a behavior called `focus1`. The behavior defines that when a focusitem that uses the behavior gets focus, that the user cannot use the tab and tab+shift keys (these are deactivated by not coupling an action to them) to navigate between text areas (all form elements are automatically focusgroup and focusroot elements). Instead, you must use the left and right arrow keys.

```
<s:behavior b:name="focus1">
    <s:initatt b:focusitem="true" b:focusindicator="true" />
    <s:whenactive>
        <s:keys b:keys="right" b:action="focusgroup-next" />
        <s:keys b:keys="left" b:action="focusgroup-previous" />
        <s:keys b:keys="tab" b:bubble="false" />
        <s:keys b:keys="tab+shift" b:bubble="false" />
    </s:whenactive>
</s:behavior>
<div>
    <input type="checkbox" b:behavior="focus1" />
    <input type="checkbox" b:behavior="focus1" />
    <input type="checkbox" b:behavior="focus1" />
</div>
```

## s:loading

The `s:loading` tag is used for setting the HTML to be displayed when the application is in the process of loading a response file. When the response file finishes loading, the HTML is automatically removed from the document.

Set the `b:controlpath` on the `body` element to specify the controls you want to use.

## Implementation

The following examples illustrates an application of the `s:loading` tag: in the process of

loading some response file, we display the "Please wait, loading data..." message in the web page.

```
<s:loading>
  <div style="position:absolute;left:50%;top:50%;">Please wait, loading data...</div>
</s:loading>
```

## s:lock

The `s:lock` tag is used for thread locking. By default, when an event is triggered, it runs asynchronously in a separate thread. Locking allows you to prevent multiple threads executing on a targeted element; when events are triggered on the element, they are forced to execute synchronously.

Locking takes place on a targeted element. You can lock any element in the document tree using the `b:target` attribute (by default, the contextnode). In other words, locking ensures that events triggered on a locked element are executed synchronously: only one execution thread can be run on the locked target - each thread must complete before the next one is started.

### Child Tags

`s:choose` , `s:fxstyle` , `s:history` , `s:if` , `s:lock` , `s:parallel` , `s:render` , `s:script` , `s:sequential` , `s:setatt` , `s:setstyle` , `s:super` , `s:task` , `s:variable` , `s:with`

### Parent Tags [Required]

`s:bookmark` , `s:event` , `s:history` , `s:httpcode` , `s:if` , `s:keys` , `s:lock` , `s:otherwise` , `s:parallel` , `s:sequential` , `s:tasklist` , `s:when` , `s:with`

### Attributes

#### `b:target`

Specify an XPath expression pointing to the target element. The default value is `.` (self).

## Implementation

In the following example, when you click the "Move rows to the right (locked)" link, it triggers the `moveright` event that targets the table rows in the first table. Each row in the table triggers the `moveright` event handler, defined in the behavior `lock-move`. The event handler locks the `tbody` element (relative XPath targeting, starting from the context node on which the event is triggered), and each thread runs synchronously. A thread must finish processing before the next one begins. When you trigger the `moveleft` event handler, exactly the same happens as in the `moveright` event handler: for each row, a separate thread is created to handle the event. However, because no lock is specified, the execution of each event is asynchronous - all the events are processed at the same time.

```
<s:behavior b:name="lock-move">
  <s:event b:on="moveright">
    <s:lock b:target="..">
      <s:task b:action="move" b:source=".." b:destination="..../~+table[1]/tbody"
b:mode="aslastchild" />
      <s:fxstyle b:cancel="false" b:time="500" />
    </s:lock>
  </s:event>
  <s:event b:on="moveleft">
    <s:task b:action="move" b:source=".." b:destination="..../~-table[1]/tbody"
b:mode="aslastchild" />
    <s:fxstyle b:cancel="false" b:time="500" />
  </s:event>
</s:behavior>
<a b:action="trigger" b:event="moveright" b:target="~+table[1]/tbody/tr" style="position:
absolute;">Move rows to the right (locked)</a>
```

```

<a b:action="trigger" b:event="moveleft" b:target="~+table[2]/tbody/tr" style="position: absolute; left:300px;">Move rows to the left (not locked)</a>
<table class="filmTable" style="position: absolute; top:25px;">
    <thead>
        <tr>
            <th style="width: 150px;">Movie</th>
            <th style="width: 60px;">Genre</th>
        </tr>
    </thead>
    <tbody>
        <tr b:behavior="lock-move">
            <td>The Seven Samurai</td>
            <td>Drama</td>
        </tr>
        <tr b:behavior="lock-move">
            <td>City of God</td>
            <td>Drama</td>
        </tr>
        <tr b:behavior="lock-move">
            <td>Donnie Darko</td>
            <td>Drama</td>
        </tr>
        <tr b:behavior="lock-move">
            <td>The Godfather</td>
            <td>Crime</td>
        </tr>
    </tbody>
</table>
<table class="filmTable" style="position: absolute; left: 300px; top:25px;">
    <thead>
        <tr>
            <th style="width: 150px;">Movie</th>
            <th style="width: 60px;">Genre</th>
        </tr>
    </thead>
    <tbody />
</table>

```

## s:otherwise

Contains the default action in an `s:choose` list. If none of the `s:when` tests return `true`, the tasks within this element will be executed. You must define the action to be taken within an `s:task` element.

```

<s:otherwise>
    <s:task b:action="commandName" />
</s:otherwise>

```

The shorthand notation for defining commands, using the `b:action` attribute, is not supported for `s:when`, `s:otherwise`, and `s:if`. For example, the construction `s:otherwise b:action="alert" b:value="Hi"` is not allowed.

### Child Tags

`s:apply-templates` , `s:attribute` , `s:call-template` , `s:choose` , `s:copy-of` ,  
`s:element` , `s:for-each` , `s:fxstyle` , `s:history` , `s:if` , `s:lock` , `s:parallel` , `s:render` ,  
`s:script` , `s:sequential` , `s:setatt` , `s:setstyle` , `s:super` , `s:task` , `s:value-of` ,  
`s:variable` , `s:with`

### Parent Tags [Required]

`s:choose`

## s:parallel

Indicate that the child elements of an `s:parallel` tag are executed in parallel (simultaneously).

### Attributes

`b:async`

Specify that the commands within a task will run asynchronously. The default is `false`. If set to `true`, a tag extended with this attribute tells the BPC engine that the response of the containing action can be ignored for the rest of the process.

Values:  
`false`  
`true`

#### Child Tags [Required]

`s:choose` , `s:fxstyle` , `s:history` , `s:if` , `s:lock` , `s:parallel` , `s:render` , `s:script` , `s:sequential` , `s:setatt` , `s:setstyle` , `s:super` , `s:task` , `s:variable` , `s:with`

#### Parent Tags [Required]

`s:bookmark` , `s:event` , `s:history` , `s:httpcode` , `s:if` , `s:keys` , `s:lock` , `s:otherwise` , `s:parallel` , `s:sequential` , `s:tasklist` , `s:when` , `s:with`

## s:render

Copies its contents and inserts the node-set at a location defined by the `b:destination` attribute. The `b:mode` attribute defines how the node-set is inserted relative to the receiving element (the node-set can also replace the receiving element).

All attributes on child nodes of the `s:render` tag are processed and checked for XPath expressions. When an XPath expression is encountered, the result of the expression is copied to the new location. To indicate that an attribute contains an XPath expression that requires resolving, surround the expression with curly brackets, for example `b:value=" {@id} "`. Note also that you can use double curly brackets `{}{xxx}` to pass the inner curly brackets as a String in the rendered content; the XPath expression is then resolved in the context of the rendered content, and not within the context of the element that triggered the render action.

The `s:render` tag can also be placed outside of an `s:event` element, in which case it must have a `b:name` attribute; you can then call it using the `render` command.

White space between elements defined within the `<s:render>` and `</s:render>` results in additional render actions, affecting performance.

#### Attributes

##### [Required] `b:destination`

Specify an XPath expression pointing to the receiving/related element. The `b:destination` of a `load`, `xsl-transform`, `send`, or `submit` command can be a variable, for example `b:destination="$myvar"`, in which case the variable is resolved to a String, or it can target an element in the BXML tree, in which case it is resolved to BXML.

##### `b:mode`

Define where a node is inserted relative to the receiving element. By default, the value is set to `replacechildren`.

Values:  
`after`  
`asfirstchild`  
`aslastchild`  
`before`  
`replace`  
`replacechildren`

##### `b:name`

Specify a logical unique name with which to identify an element.

##### `b:test`

Specify a condition, expressed in XPath, that returns either true or false. For example, `b:test="@border = '0'"` checks whether the border attribute of the context node is set to 0; if it isn't, the code is not executed.

#### Child Tags

You can define any BXML (and HTML) elements as children of this tag. You can also use the following tag-specific child tags:

**s:copy-of**, **s:textnode**

#### Parent Tags

**s:bookmark**, **s:event**, **s:history**, **s:httpcode**, **s:if**, **s:keys**, **s:lock**, **s:otherwise**, **s:parallel**, **s:sequential**, **s:tasklist**, **s:when**, **s:with**

### Implementation

In the following example, each time you click the "Item" link, the child elements of the **s:render** tag are copied to the target empty **div** tag. The number of copies depends on the number of commands issued by the mouse click.

Every copied "Item" is placed as the last child in the **div** node-set. The "Clear copied items" link contains a **remove** command, which allows you to **undo** the copy action by emptying the contents of the targeted **div** tag.

```
<a>
    <s:event b:on="command">
        <s:render b:destination="id('copyTarget')" b:mode="aslastchild">
            <div>Item</div>
        </s:render>
    </s:event>
    Copy "item" to the box.
</a>


In the following example, a window is rendered when you click the button and the content of the id attribute is inserted in the text node of the body element. The example is interesting because it demonstrates the use of a double curly brackets b:value="{{@id}}" in a render action. Single curly brackets {xxx} instruct the BPC to resolve XPath expression during the render phase and return the result as a String. Double curly brackets { {xxx} } instruct the BPC to pass the contents of the outer curly brackets ({xxx}) as a String in the rendered content; the XPath expression is then resolved in the context of the rendered content, and not within the context of the element that triggered the render action. Therefore, using single curly brackets {@id} in the code below would result in the text theButton being inserted into the window body instead of theWindow (the result of {{{@id}}}).



Backbase is similar to XSLT in using single curly brackets {xxx} to indicate that the attribute value contains an XPath expression that requires resolving. It differs from XSLT, however, in that the value must start with the opening curly { bracket. If you want to concatenate strings, use the concat() function.



```
<b:button id="theButton">The Button
    <s:event b:on="command">
        <s:render b:destination="id('window-main')" b:mode="replace">
            <b>window id="theWindow">
                <s:event b:on="construct">
                    <s:task b:action="set" b:target="id('div')/text()" b:value="{{{@id}}}" />
                </s:event>
                <b>windowhead />
                <b>windowbody>
                    <div id="div">id attribute inserted here</div>
                </b>windowbody>
            </b>window>
        </s:render>
    </s:event>
```



©2004-2006 Backbase B.V., All Rights Reserved.



35


```

```
</b:button>
<div id="window-main"></div>
```

## s:script

The `s:script` tag allows to add inline JavaScript in your BXML. The difference between the `s:script` and `script` elements is the scope: in the former, the functions you have defined are available locally within the element in which the `s:script` element is defined, in the latter, functions are available globally throughout your application.

An advantage of using inline JavaScript is that you have all the Backbase variables (local, global, and tag) at your disposal that you also have in XPath. You can access the variables using the syntax `_vars['myvarname'][0]`. The variables are nodelists and are represented as an array of values. Even if the nodelist only contains one value, you have to retrieve it from the array, using for example `[0]`. You can also get the current node using the `_current` variable.

When defined within an execution element, such as an `s:event` tag, the `s:script` is executed sequentially. For example:

```
<s:event b:on="command">
  <s:task b:action="alert" b:value="executed first" />
  <s:script>
    <!--executed second-->
</s:script>
  <s:task b:action="alert" b:value="executed third" />
</s:event>
```

You must always properly escape the inline JavaScript contained in the `s:script` tag otherwise the characters within your code causes XML parse errors. It is recommended to encapsulate the script within `s:script` tags using XML comments.

### Parent Tags [Required]

- `s:bookmark`
- `s:event`
- `s:history`
- `s:httpcode`
- `s:if`
- `s:keys`
- `s:lock`
- `s:otherwise`
- `s:parallel`
- `s:sequential`
- `s:tasklist`
- `s:when`
- `s:with`

## Implementation

The following is an example of defining inline JavaScript:

```
<div>
  <s:event b:on="construct">
    <s:variable b:name="myvar" b:select=".." b:scope="local" />
    <s:script>
      <!-- _vars['myvar'][0].style.backgroundColor = 'blue'; _current.style.color =
'white';-->
    </s:script>
  </s:event>
  This text will be white on a blue background
</div>
```

If your script contains XML sensitive characters such as `--`, you must escape using a different syntax. For example, using the syntax `alert(i--)` leads to parsing errors; instead, use `alert(i=i-1)` or `alert(i-=1)`.

```
<span>
  <s:event b:on="command">
    <s:script>
      <!-- var i=3; alert(i-=1); alert(i=i-1); -->
    </s:script>
  </s:event>
  Click to see alert with variable value
</span>
```

## s:sequential

The execution tags, which are the child elements of the `s:sequential` tag such as `s:fxstyle` and `s:task` for example, will be executed in a sequential fashion, one after another. By default, all transition effects run sequentially, which means that this tag may be omitted.

### Attributes

#### b:async

Specify that the commands within a task will run asynchronously. The default is `false`. If set to `true`, a tag extended with this attribute tells the BPC engine that the response of the containing action can be ignored for the rest of the process.

##### Values:

`false`  
`true`

### Child Tags [Required]

`s:choose` , `s:fxstyle` , `s:history` , `s:if` , `s:lock` , `s:parallel` , `s:render` , `s:script` , `s:sequential` , `s:setatt` , `s:setstyle` , `s:super` , `s:task` , `s:variable` , `s:with`

### Parent Tags [Required]

`s:bookmark` , `s:event` , `s:history` , `s:httpcode` , `s:if` , `s:keys` , `s:lock` , `s:otherwise` , `s:parallel` , `s:sequential` , `s:tasklist` , `s:when` , `s:with`

## s:setatt

Use this tag to set multiple attributes on an element. You can set any attribute using this technique. You can use the `s:with` tag to set attributes on elements other than the current contextnode.

### Implementation

```
<div>
  <s:event b:on="command">
    <s:setatt style="background-color: red; color: white;" />
  </s:event>
  CLICK THIS TEXT
</div>
```

### Parent Tags [Required]

`s:bookmark` , `s:event` , `s:history` , `s:httpcode` , `s:if` , `s:keys` , `s:lock` , `s:otherwise` , `s:parallel` , `s:sequential` , `s:tasklist` , `s:when` , `s:with`

## s:setstyle

Use this tag to set styles of a target element. The target element corresponds to the names of the `b:attribute` used. For example, `b:background` sets the background style property. The syntax and notation for setting styles is the same as when using CSS, except a `b:` precedes the attribute.

### Attributes

#### b:background

A shorthand property for setting the individual background properties (i.e., `background-color`, `background-image`, `background-repeat`, `background-attachment` and `background-position`) at the same place in the style sheet.

#### b:background-attachment

If a background image is specified, this property specifies whether it is fixed with regard to the viewport (`fixed`) or scrolls along with the containing block (`scroll`). `inherit` means that, for a given element, the property takes the same computed value as the property for the element's parent.

##### Values:

`fixed`

```
inherit
scroll
```

**b:background-color**

Sets the background color of an element, either a `color` value or the keyword `transparent` to make the underlying colors shine through.

Values:

```
<color>
inherit
transparent
```

**b:background-image**

Sets the background image of an element using a URI. You should also specify a background color that is used when the image is unavailable. When the image is available, it is rendered on top of the background color. (Thus, the color is visible in the transparent parts of the image).

Values:

```
<uri>
inherit
none
```

**b:background-position**

If a background image has been specified, this property specifies its initial position.

Values:

```
<length>
<percentage>
center
left
right
```

**b:background-repeat**

If a background image is specified, this property specifies whether the image is repeated (tiled), and how. All tiling covers the content, padding and border areas of a box.

Values:

```
inherit
no-repeat
repeat
repeat-x
repeat-y
```

**b:border-collapse**

Set the border model used on table cells. The value `separate` selects the separated borders border model, for separated borders around individual cells; the value `collapse` selects the collapsing borders model for borders that are continuous from one end of the table to the other.

Values:

```
collapse
inherit
separate
```

**b:border-color**

Specify the color of a box's border. Border values `{1,4}` refer respectively to top, right, bottom, and left border. For example, `{border-color: red green blue}` results in red top border, green left and right border, and, blue bottom border. `{border-color: red green blue yellow}` results in red top and bottom border, and green left and right border.

Values:

```
<color> | transparent | {1,4} | inherit
```

**b:border-style**

Specify the line style of a box's border. All borders are drawn on top of the box's background. The color of borders depends on the element's border color properties. You can specify `{1-4}` values, indicating top-right-bottom-left values for each border edge. When you want to set the border styles for just one side of an element, rather than all four, use single-side styles; `border-top-style`, `border-right-style`, `border-bottom-style`, `border-left-style`.

- `none` - no border
- `hidden` - same as `none`, except in terms of border conflict resolution for ta-

ble elements

- dotted - a series of dots
- dashed - a series of short line segments
- double - two solid lines. The sum of the two lines and the space between them equals the value of border-width
- groove - the border looks as though it is carved into the canvas
- ridge - the opposite of groove: the border looks as though it were coming out of the canvas
- inset - the box look as though it were embedded in the canvas
- outset - the opposite of inset: the border makes the box look as though it were coming out of the canvas

Values:

dashed  
dotted  
double  
groove  
hidden  
inherit  
inset  
none  
outset  
ridge

#### b:border-top

A shorthand property for setting the width, style, and color of the top border of a box.

Values:

[ <border-width> || <border-style> || <border-top-color> ]  
inherit

#### b:border-right

A shorthand property for setting the width, style, and color of the right border of a box.

#### b:border-bottom

A shorthand property for setting the width, style, and color of the bottom border of a box.

#### b:border-left

A shorthand property for setting the width, style, and color of the left border of a box.

#### b:border-top-color

Specify the color of the top edge of a box's border.

Values:

<color>  
inherit  
transparent

#### b:border-right-color

Specify the color of the right edge of a box's border.

Values:

<color>  
inherit  
transparent

#### b:border-bottom-color

Specify the color of the bottom edge of a box's border.

Values:

<color>  
inherit  
transparent

#### b:border-left-color

Specify the color of the left edge of a box's border.

Values:

<color>  
inherit  
transparent

#### b:border-top-style

Specify the line style of the top edge of a box's border.

Values:

dashed  
dotted  
double  
groove  
hidden  
inherit  
inset  
none  
outset  
ridge

**b:border-right-style**

Specify the line style of the right edge of a box's border.

Values:  
dashed  
dotted  
double  
groove  
hidden  
inherit  
inset  
none  
outset  
ridge

**b:border-bottom-style**

Specify the line style of the bottom edge of a box's border.

Values:  
dashed  
dotted  
double  
groove  
hidden  
inherit  
inset  
none  
outset  
ridge

**b:border-left-style**

Specify the line style of the left edge of a box's border.

Values:  
dashed  
dotted  
double  
groove  
hidden  
inherit  
inset  
none  
outset  
ridge

**b:border-top-width**

Specify the width of the top edge of a box's border.

Values:  
<length>  
inherit  
medium  
thick  
thin

**b:border-right-width**

Specify the width of the right edge of a box's border.

Values:  
<length>  
inherit  
medium  
thick  
thin

**b:border-bottom-width**

Specify the width of the bottom edge of a box's border.

Values:  
<length>  
inherit  
medium  
thick

thin

#### b:border-left-width

Specify the width of the left edge of a box's border.

#### b:border-width

A shorthand property used for setting the width of the border area; border-top-width, border-right-width, border-bottom-width, and border-left-width. If there is only one value, it applies to all sides. If there are two values, the top and bottom borders are set to the first value and the right and left are set to the second. If there are three values, the top is set to the first value, the left and right are set to the second, and the bottom is set to the third. If there are four values, they apply to the top, right, bottom, and left, respectively. (If you specify a length, set an explicit value (must not be negative.)

Values:

<length>  
inherit  
medium  
thick  
thin

#### b:bottom

A percentage or pixel value that sets the bottom position from the bottom edge of the containing element. Negative values are allowed.

#### b:clear

Indicate which sides of an element's box(es) may not be adjacent to an earlier floating box. [clear](#) does not consider floats inside the element itself or in other block formatting contexts.

Values:

both  
left  
none  
right

#### b:color

Describe the foreground color of an element's text content.

Values:

<color>  
inherit

#### b:cursor

Control the type of cursor that should be used for a specific section of the site.

Typically, if you use a [div](#) tag with a [b:action](#) attribute, you also want to assign the cursor pointer to inform the user that the element is clickable.

You can, for example, set a default cursor for the entire web application on the first content node in your web application. This suppresses the browser's default behavior to change cursors when hovering over normal text, therefore resulting in a more consistent application.

The cursor appearance can vary from browser to browser.

Values:

crosshair  
default  
e-resize  
help  
move  
ne-resize  
n-resize  
nw-resize  
pointer  
text  
wait

#### b:direction

Specify the base writing direction of blocks (left-to-right, or right-to-left) and the direction of embeddings and overrides (see 'unicode-bidi') for the Unicode bidirectional algorithm. In addition, it specifies the direction of table column layout, the direction of horizontal overflow, and the position of an incomplete last line in a block in case of [text-align: justify](#).

Values:

inherit

ltr  
rtl

**b:display**

Define the kind of display box an element generates during layout.

Values:  
block  
inherit  
inline  
inline-block  
inline-table  
list-item  
none  
run-in  
table  
table-caption  
table-cell  
table-column  
table-column-group  
table-footer-group  
table-header-group  
table-row  
table-row-group

**b:float**

Define the direction in which an element is floated.

Values:  
inherit  
left  
none  
right

**b:font**

A shorthand property to set multiple font-related attributes with one assignment statement. For example, {font: 12px serif} .

Values:  
<font-style> [ <font-style> || <font-variant> || <font-weight>> ]? <font-size> [ / <line-height> ]? <font-family> ]  
caption  
icon  
inherit  
menu  
message-box  
small-caption  
status-bar

**b:font-family**

Define a font family to be used in the display of an element's text.

Values:  
[ [ <family-name> | <generic-family> ] [ , <family-name> | <generic-family> ]\* ] ]  
inherit

**b:font-size**

Set the size of the font for an element.

Values:  
<length>  
<percentage>  
inherit  
large  
larger  
medium  
small  
smaller  
x-large  
x-small  
xx-large  
xx-small

**b:font-style**

Set the font to use italic, oblique, or normal face font.

Values:  
inherit  
italic  
normal  
oblique

**b:font-variant**

Define small-caps text.

**Values:**  
 inherit  
 normal  
 small-caps

### b:font-weight

Set the font weight used in rendering the element's text.

**Values:**  
 100  
 200  
 300  
 400  
 500  
 600  
 700  
 800  
 900  
 bold  
 bolder  
 inherit  
 lighter  
 normal

### b:height

Specify a percentage or pixel value for the height of an element.

### b:left

Specify a percentage or pixel value that sets the left position from the left edge of the containing element. Negative values are allowed.

### b:letter-spacing

Define the amount of whitespace to be inserted between the character boxes of text.

**Values:**  
 <length>  
 inherit  
 normal

### b:line-height

Influence the layout of line boxes.

**Values:**  
 <length>  
 <number>  
 <percentage>  
 inherit  
 normal

### b:list-style

A shorthand property that condenses all other [list-style](#) properties.

**Values:**  
 [ <list-style-type> || <list-style-position> ||  
 <list-style-image> ]  
 inherit

### b:list-style-image

Specify an image to be used as the marker on an ordered or unordered list item.

**Values:**  
 <uri>  
 inherit  
 none

### b:list-style-position

Declare the position of the list marker with respect to the content of the list item.

**Values:**  
 inherit  
 inside  
 outside

### b:list-style-type

Declare the type of the marker system to be used in the presentation of a list (CSS2.1 values).

**Values:**  
 armenian  
 circle

```
decimal
decimal-leading-zero
disc
georgian
inherit
lower-greek
lower-latin
lower-roman
none
square
upper-latin
upper-roman
```

**b:margin**

A shorthand property that sets the margin for all four sides. {1,4} refers to respectively margin-top, margin-right, margin-bottom, and margin-left. (The other margin properties only set their respective side.)

**Values:**

```
[<length> | <percentage> | auto ] {1,4}
inherit
```

**b:margin-top**

Set the width of a top margin for an element.

**Values:**

```
<length>
<percentage>
auto
inherit
```

**b:margin-right**

Set the width of a right margin for an element.

**Values:**

```
<length>
<percentage>
auto
inherit
```

**b:margin-bottom**

Set the width of a bottom margin for an element.

**Values:**

```
<length>
<percentage>
auto
inherit
```

**b:margin-left**

Set the width of a left margin for an element.

**Values:**

```
<length>
<percentage>
auto
inherit
```

**b:overflow**

Specify whether content of a block-level element is clipped when it overflows the element's box.

**Values:**

```
auto
hidden
inherit
scroll
visible
```

**b:opacity**

Specify the opacity of an element as a number between 0 and 100.

*Note: Using opacity with transitions ([fxstyle](#)) can significantly reduce performance, therefore use sparingly.*

**b:padding**

A shorthand property for setting all padding properties. {1,4} refers respectively to padding-top, padding-right, padding-bottom, and padding-left. For example, {padding: 0.5cm 2.5cm} sets a top and bottom padding of 0.5cm, and a left and right padding of 2.5cm.

**Values:**

```
[<length> | <percentage>] {1,4}
```

inherit  
**b:padding-top**

Set the width of the top padding for an element.

Values:  
 <length>  
 <percentage>  
 inherit

**b:padding-right**

Set the width of the right padding for an element.

Values:  
 <length>  
 <percentage>  
 inherit

**b:padding-bottom**

Set the width of the bottom padding for an element.

Values:  
 <length>  
 <percentage>  
 inherit

**b:padding-left**

Set the width of the left padding for an element.

Values:  
 <length>  
 <percentage>  
 inherit

**b:position**

Define the positioning scheme used to layout an element.

Values:  
 absolute  
 fixed  
 inherit  
 relative  
 static

**b:right**

Specify a percentage or pixel value that sets the right position from the right edge of the containing element. Negative values are allowed.

**b:text-align**

Describe how inline content of a block is aligned (CSS2.1 values).

Values:  
 center  
 inherit  
 justify  
 left  
 right

**b:text-decoration**

Describe a decoration added to the text. A line can be added; under, above, or through the text. The blink value defines a blinking text.

Values:  
 [ underline || overline || line-through || blink ]  
 inherit  
 none

**b:text-indent**

Define the indentation of the first line of text in a block-level element.

Values:  
 <length>  
 <percentage>  
 inherit

**b:text-transform**

Change the case of letters in an element, regardless of the case of the text in the document source.

Values:  
 capitalize  
 inherit  
 lowercase  
 none  
 uppercase

**b:top**

Specify a percentage or pixel value that sets the top position from the top edge of the containing element. Negative values are allowed.

#### b:vertical-align

Specify the vertical positioning inside a line box of the boxes generated by an inline-level element.

Values:

```
<length>
<percentage>
baseline
bottom
inherit
middle
sub
super
text-bottom
text-top
top
```

#### b:visibility

Specify whether the boxes generated by an element are rendered.

Values:

```
collapse
hidden
inherit
visible
```

#### b:white-space

Declare how whitespace within an element is handled during layout.

Values:

```
inherit
normal
nowrap
pre
pre-line
pre-wrap
```

#### b:width

Specify a percentage or pixel value specifying the width of an element.

#### b:word-spacing

Declare how whitespace inside the element is handled.

Values:

```
inherit
normal
nowrap
pre
pre-line
pre-wrap
```

#### b:z-index

Specify, for a positioned box, the stack level of the box in the current stacking context, and whether the box establishes a local stacking context.

Values:

```
<integer>
auto
inherit
```

---

#### Child Tags

Child tags are not allowed in this tag.

---

#### Parent Tags [Required]

**s:bookmark , s:event , s:history , s:httpcode , s:if , s:keys , s:lock , s:otherwise , s:parallel , s:sequential , s:tasklist , s:when , s:with**

## Implementation

The following example sets the color and background-color of the current element.

```
<div>
  <s:event b:on="command">
    <s:setstyle b:background-color="red" b:color="white" />
  </s:event>
  CLICK THIS TEXT
</div>
```

## s:state

You can use `s:state` within a `s:behavior` tag to define the visual representation of an element when in one of its base states. An element can be in the state **selected**, or in the state **deselected**. The `s:state` attaches a different CSS class to an element depending on its state. For each of these states, you can specify CSS classes for when the mouse hovers over the element, for when the element is pressed, for when the element is disabled, and for the element's initial "normal" appearance.

The CSS classes that are applied to an element using the `s:state` tag are concatenated to existing classes applied to an element.

Note: At any one time, an element in a Backbase application can be in a state **selected**, or in a state **deselected**. You can specify the default state of an element using the `b:state` attribute.

### Attributes

---

#### [Required] `b:on`

Attaches a style class to an element depending on its state.

Values:  
deselect  
select

#### [Required] `b:normal`

Specify the style class(es) to use when the element is in its base state.

#### `b:hover`

Specify the style class to use when the mouse hovers over the element (corresponds to CSS `:hover` and the `mouseover` event).

#### `b:press`

Specify the style class to use when the element is being pressed (corresponds to the CSS psuedo class `:active`, and the `mousedown` event).

#### `b:disabled`

Disable an element. No user input events can be triggered on an element that is disabled, only system events. You can enable a disabled element using the `set` or `enable` commands, or by using the `s:setatt` tag.

---

### Child Tags

Child tags are not allowed in this tag.

---

### Parent Tags [Required]

#### `s:behavior`

---

## Implementation

In the following example, the behavior defines different CSS classes to use when an element is in one of the basic states selected or deselected, and for each state the appearance when the mouse hovers over or clicks the element.

Additional note: you can use the same principle of HTML class attributes - multiple CSS classes can be assigned from within one attribute with a space separated list of values.

```
<div b:behavior="statebehavior">      Click or hover here to see the
  <code>s:state</code>
tag in action.
</div>
<s:behavior b:name="statebehavior">
  <s:state b:on="deselect" b:normal="sb" b:hover="sb sb-hover" b:press="sb sb-press" />
  <s:state b:on="select" b:normal="sb sb-selected" b:hover="sb sb-selected sb-hover"
b:press="sb sb-selected sb-press" />
  <s:event b:on="command" b:action="select-deselect" />
</s:behavior>
<style type="text/css">      .sb {      border: 2px solid #889;      padding: .3em;
cursor: pointer;      }      .sb-selected {      background: blue;      color: white;      }
      .sb-hover {      background: red;      }      .sb-press {      background: yellow;
}      </style>
```

## s:stylesheet

The `s:stylesheet` template mechanism tag defines the starting point of a block of processing/transformation rules. It can be located anywhere in a web application's files, as long as it is available at the time it is invoked.

### Attributes

#### [Required] b:name

Specify the name of a template.

### Child Tags [Required]

#### [Required] s:template

## s:super

Allows a behavior to inherit the events defined in another behavior. The BPC engine first goes to the inherited behavior and executes the event handler before proceeding with the current behavior. The command after the `s:super` tag is processed when the execution of the inherited behavior (for the same event) has completed.

Behavior inheritance operates only when an event handler is defined in the inherited behavior for the **same** event.

### Parent Tags [Required]

#### s:event , s;if , s:lock , s:otherwise , s:parallel , s:sequential , s:when

## Implementation

In the following example, the behavior called `movedown` defines an event handler for moving objects down in the object hierarchy. It inherits the `zoom` behavior that handles the visual style changes. The `s:super` tag makes sure that the event of the same type in the inherited behavior is applied before starting to run the core behavior instructions for that event. Note that both behaviors define an event handler for the `click` event.

```
<s:behavior b:name="zoom">
    <s:event b:on="click">
        <s:fxstyle b:font-size="15px" />
        <s:fxstyle b:background-color="#B8B8B8" />
        <s:fxstyle b:font-size="20px" />
        <s:fxstyle b:background-color="#888888" />
    </s:event>
</s:behavior>
<s:behavior b:name="movedown" b:behavior="zoom">
    <s:event b:on="click">
        <s:super />
        <s:task b:action="movedown" b:target=". ." />
        <s:task b:action="show-hide" b:target="span[@at='moveddown']" />
    </s:event>
</s:behavior>
<div>
    <p style="background-color: #F0F0F0; font-size=10;" b:behavior="movedown">Life is
Beautiful &gt;&gt;
    <span at="moveddown">is an Italian language film which tells the story of an Italian
Jew who lives in a romantic fairy tale, but must learn how to use that dreamy quality to
survive a concentration camp with his young son Joshua.</span>
</p>
    <p style="background-color: #F0F0F0; font-size=10;" b:behavior="movedown">2001: A Space
Odyssey &gt;&gt;
    <span at="moveddown">is an influential science fiction film notable for combining
episodes contrasting high levels of scientific and technical realism with transcendental
mysticism.</span>
</p>
    <p style="background-color: #F0F0F0; font-size=10;" b:behavior="movedown">The Seven
Samurai &gt;&gt;
    <span at="moveddown">...takes place in war-ridden 16th century Japan, where a village
of farmers look for ways to ward off a band of marauding robbers.</span>
</p>
</div>
```

## s:task

This tag implements the command specified in the `b:action` attribute. The specified command determines the attributes that you need to place on the tag to execute the command. For more information, see the command section.

### Attributes

#### [Required] `b:action`

Specify the command to execute when the element's command event fires, for example when it is clicked. Each command has specific attributes. See the command section for an overview of all the commands.

#### `b:test`

Specify a condition, expressed in XPath, that returns either true or false. For example, `b:test="@border = '0'"` checks whether the border attribute of the context node is set to 0; if it isn't, the code is not executed.

#### `b:async`

Specify that the commands within a task will run asynchronously. The default is `false`. If set to `true`, a tag extended with this attribute tells the BPC engine that the response of the containing action can be ignored for the rest of the process.

Values:  
`false`  
`true`

### Child Tags

Child tags are not allowed in this tag.

### Parent Tags [Required]

`s:bookmark` , `s:event` , `s:history` , `s:httpcode` , `s:if` , `s:keys` , `s:lock` , `s:otherwise` , `s:parallel` , `s:sequential` , `s:tasklist` , `s:when` , `s:with`

## s:tasklist

A container for a sequence of commands. This tag contains a list of `s:task` tags that are executed when the tasklist is called. You can call the tasklist from an *event handler* using the syntax `b:on<EventName>=<TaskListName>`, for example `b:onclick="mytasklist"` calls the tasklist called `mytasklist` when the `click` event is triggered.

### Attributes

#### [Required] `b:name`

Specify a logical unique name with which to identify an element.

### Child Tags [Required]

`s:choose` , `s:fxstyle` , `s:history` , `s:if` , `s:lock` , `s:parallel` , `s:render` , `s:script` , `s:sequential` , `s:setatt` , `s:setstyle` , `s:task` , `s:variable` , `s:with`

## Implementation

The link element in this example is extended with an event handler (in this case, the `b:onclick` event). When clicking on this element the tasklist defined in the event handler will be executed. In this case it will show an alert box.

```
<a b:onclick="tasklist_ex">Click here to run the tasklist.</a>
<s:tasklist b:name="tasklist_ex">
  <s:task b:action="alert" b:value="Hello!" />
</s:tasklist>
```

## s:template

The `s:template` template mechanism tag contains the rules to apply when a specified node is matched.

---

## Attributes

**[Required] b:match**

Associate the template with an XML element. It can also be used to define a template for the whole branch of an XML document. Specify a name, or "/" to denote the root template. "\*" is an XPath expression to denote all elements.

**b:preserve whitespace**

Set to `true` to preserve the whitespaces defined in the `s:template` tag. Set to `false` to remove whitespaces in textnodes (the default).

**Child Tags [Required]**

`s:apply-templates` , `s:attribute` , `s:call-template` , `s:choose` , `s:copy-of` , `s:element` , `s:for-each` , `s:if` , `s:value-of` , `s:variable`

**Parent Tags [Required]**

**[Required] s:stylesheet**

---

## s:textnode

Used inside the `s:render` tag to create text nodes. The `b:label` attribute contains the text to place as a textnode.

**Attributes****[Required] b:label**

A string value that is inserted as a textnode child of the parent of the `s:textnode` tag.

**Parent Tags [Required]**

`s:render`

---

## s:tooltip

The `s:tooltip` tag can be used to create advanced *tooltips*. Anything you place inside the tooltip tag will be used as the content of the tooltip. If you want a less advanced tooltip, you can simply add the `b:tooltiptext` attribute to your element.

Set the `b:controlpath` on the `body` element to specify the controls you want to use.

### Implementation

In the following example, when you move the mouse over the `span` element, the tooltip is displayed; when you move the mouse out of the `span` element, the tooltip is deselected.

```
<span style="border: 1px solid #000000;padding:2px;">Hover here to show tooltip.
  <s:tooltip>
    <strong>This is a tooltip.</strong>
    <br />
    You can put all kinds of
    <em>text</em>
  in here.
  </s:tooltip>
</span>
```

---

It is important to note that, in IE, HTML block-level elements such as `div` elements can not be placed inside `p` tags. Doing this divides the `p` in two block-level elements and produces unexpected results. For more information, read [Browser Issues - Technical Article \[Backbase Manual\]](#).

## s:value-of

The `s:value-of` tag extracts the value of the selected node or attribute and inserts it into the htmlstructure as a text node.

**Attributes**

**[Required] b:select**

Specify the element that is selected using an XPath statement.

**Parent Tags [Required]****s:htmlstructure****s:value-of**

The `s:value-of` tag extracts the value of the selected node or attribute and inserts it into the output of the transformation.

**Attributes****[Required] b:select**

Specify the element that is selected using an XPath statement.

**Parent Tags [Required]****s:template****s:variable**

Container to define a variable. A variable is an item of data (`b:select`) named by an identifier (`b:name`) and has a scope (`b:scope`) that determines where the variable can be used. The scope of the variable can be global, local, or tag. You can define the variable inside any event handler.

You can access the variable from XPath using the `$` dollar sign and the name of the variable, for example `$myvar`, or from JavaScript using the `_vars['myvars'][0]` variable (in which case you DO NOT need to use the `$` sign). You can set a new value for the variable using the `assign` command.

A variable is a reference (a pointer) to a node-set; assigning a new value to the variable therefore updates the node-set referred to in the variable. Conflicts between variable names, defined using different scopes, are resolved in the order of importance: `tag`, `local`, `global`.

**Attributes****[Required] b:name**

Specify a logical unique name with which to identify an element.

**[Required] b:select**

Specify the element that is selected using an XPath statement.

**b:scope**

Specify where the variable can be used:

A `local` variable (the default) is available within the scope of the element in which is triggered, and is passed automatically along with any event calls within the event handler. The variable is cleared from memory when the original execution thread ends.

A `global` variable is available throughout your application and exists for the lifespan of the application.

A `tag` variable is available within the scope of the element in which it is defined, and exists for the lifespan of the application.

**Values:**

`global`  
`local`  
`tag`

**Child Tags**

`s:apply-templates` , `s:attribute` , `s:call-template` , `s:choose` , `s:copy-of` ,  
`s:element` , `s:for-each` , `s:fxstyle` , `s:history` , `s:if` , `s:lock` , `s:parallel` , `s:render` ,  
`s:script` , `s:sequential` , `s:setatt` , `s:setstyle` , `s:task` , `s:value-of` , `s:variable` ,  
`s:with`

**Parent Tags**

s:bookmark , s:element , s:event , s:for-each , s:history , s:httpcode , s;if , s;if , s:keys , s:lock , s:otherwise , s:otherwise , s:parallel , s:sequential , s:tasklist , s:template , s:value-of , s:variable , s:when , s:when , s:with

## Implementation

In the following example, a `tag` variable is declared inside the `div` element; the variable is created when the page is constructed using the `construct` event handler. When you click the link element, you call the variable by targeting the `div` element in which it is defined; the value is then put in an alert box.

```
<div id="my-div" b:myatt="Hello world! ">
    <s:event b:on="construct">
        <s:variable b:name="myTag" b:scope="tag" b:select="@b:myatt" />
    </s:event>
</div>
<a>Click here
    <s:event b:on="command">
        <s:task b:action="alert" b:value="{id('my-div')/$myTag}" />
    </s:event>
</a>
```

In the following example, you declare a `global` variable inside the `construct` event handler. When you click the link element, you call the variable and put its value in an alert box.

```
<s:event b:on="construct">
    <s:variable b:name="myTag" b:scope="global" b:select="id('my-div')/@b:myatt" />
</s:event>
<div id="my-div" b:myatt="Hello world! "></div>
<a>Click here
    <s:event b:on="command">
        <s:task b:action="alert" b:value="{$myTag}" />
    </s:event>
</a>
```

In the following example, you declare a `local` variable inside the `click` event handler of the first `div` element. When you click the element, the variable is created and passed on to the event handler in the second `div`, which then puts its value in an alert box. When you close the alert box, the local variable dies because it is only available within the context of the execution thread in which it was created. Therefore, clicking the second `div` element results in an XPath resulted in an empty node-set error.

```
<div id="my-div" b:myatt="Hello world!">Click Here (success)
    <s:event b:on="click">
        <s:variable b:name="myTag" b:scope="local" b:select="@b:myatt" />
        <s:task b:action="trigger" b:event="command" b:target="id('trgl1')" />
    </s:event>
</div>
<div id="trgl1">Try Clicking Here (fail)
    <s:event b:on="command">
        <s:task b:action="alert" b:value="{$myTag}" />
    </s:event>
</div>
```

## **s:when**

Lists choices in a sequence of `s:choose` elements. Tasks within this element will only be executed if the test returns true.

The shorthand notation for defining commands (the `b:action` attribute) for `s:when`, `s:otherwise`, `s;if` is not supported. You must define the action to be taken within a `s:task` element. For example, the construction `s:when b:test="" b:action="commandName" /` is not supported; instead use the following:

```
<s:when b:test="">
  <s:task b:action="commandName" />
</s:when>
```

### Attributes

#### [Required] b:test

Specify a condition, expressed in XPath, that returns either true or false. For example, `b:test="@border = '0'"` checks whether the border attribute of the context node is set to 0; if it isn't, the code is not executed.

### Child Tags

`s:apply-templates` , `s:attribute` , `s:call-template` , `s:choose` , `s:copy-of` , `s:element` , `s:for-each` , `s:fxstyle` , `s:history` , `s:if` , `s:lock` , `s:parallel` , `s:render` , `s:script` , `s:sequential` , `s:setatt` , `s:setstyle` , `s:task` , `s:value-of` , `s:variable` , `s:with`

### Parent Tags [Required]

`s:choose`

## s:whenactive

Wrap `s:keys` in the `s:whenactive` tag to provide the application context for when the keys are active. The keys are only active when the elements to which they are applied are active. You can define `s:whenactive` inline, or inside a behavior. For example, to define global keys, you can define them in a `s:behavior` that is used in the `body` element to be active throughout your application.

An element can have one of the states active or inactive: an element is active when it, or a child node, has focus; and an element is inactive when it or a child node does not have focus (is blurred). The `active` and `inactive` events take place on elements when they change their state.

You can also use the `s:whenactive` tag to override the default navigation keys defined for the focus model.

### Child Tags [Required]

`s:keys`

### Parent Tags

`s:behavior`

## Implementation

The following example defines a behavior for input elements. The up and down arrow keys are assigned `focusitem-next` and `focusitem-previous` commands to allow you to jump to the next or previous focusable elements using the up and down arrow keys.

```
<s:behavior b:name="my-focus-model">
  <s:whenactive>
    <s:keys b:keys="up" b:action="focusitem-previous" />
    <s:keys b:keys="down" b:action="focusitem-next" />
  </s:whenactive>
</s:behavior>
<input type="text" b:behavior="my-focus-model" />
<input type="text" b:behavior="my-focus-model" />
<input type="text" b:behavior="my-focus-model" />
```

## s:with

Allows you to target an element for which you want to use the `s:setatt` and `s:setstyle` elements.

### Attributes

#### [Required] b:target

Specify an XPath expression pointing to the target element. The default value

is . (self).

#### Child Tags

s:choose , s:fxstyle , s:history , s:if , s:lock , s:parallel , s:render , s:script , s:sequential , s:setatt , s:setstyle , s:task , s:variable , s:with

#### Parent Tags [Required]

s:bookmark , s:event , s:history , s:httpcode , s:if , s:keys , s:lock , s:otherwise , s:parallel , s:sequential , s:tasklist , s:when , s:with

## S:xml

In Backbase templating, the `s:xml` tag is used to delimit the XML data source when it is part of the main BXML document. You must specify the `b:name` attribute for the `s:xml` tag. The name of the datasource is added as a global variable. The contents of the `s:xml` tag must be well-formed.

### Implementation

In the following example, you can call the XML datasource using the syntax `$catalog` (for more information, see the `transform` command and `s:stylesheet` tag):

```
<s:xml b:name="catalog">
  <products>
    <product>
      <name>Apple</name>
      <description>A Trendy Computer</description>
    </product>
    <product>
      <name>PC</name>
      <description>A Functional Computer</description>
    </product>
  </products>
</s:xml>
```

#### Attributes

##### [Required] `b:name`

Specify a logical unique name with which to identify an element.

## HTML Tag Extensions

### xhtml:body

The standard HTML `body` tag used for declaring the start of the content of a web page. You can extend the tag with BXML attributes.

#### Attributes

##### b:devconfig

A string pointing to the tool configuration xml file. If you specify `false`, the Backbase developer tools are not loaded making the startup of the Backbase application significantly faster. Note that the attribute only applies to the development edition; the tools are not available and are therefore never loaded in the production edition.

##### b:controlpath

Set a path to load controls automatically. The default path is the `controls/dynamic` folder. You set the attribute on the `body` tag.

##### b:skinbase

Specify the file to use that contains default behaviors, styling, and layout (the "skinbase") for the Backbase controls:

- `default` - loads the `default.xml` file; the `b:controlpath` attribute specifies the location from which it is loaded. The `default.xml` is loaded by default.
- `none` - prevents the loading of the `default.xml` file.
- `<relativePathToFile>` - specify a relative url pointing to the file that is loaded instead of the `default.xml`. The path to the file is relative to the value of the `b:controlpath` attribute. For example, you can load the `default_without_form.xml` that has the same content as `default.xml` but without the basic form validation logic. (When you do not use the form logic, it is recommended you use this file.)

You can use the skinbase configuration file as a central location for including miscellaneous behavior and styling. Loading the file automatically is a convenient way to ensure your applications default look and feel (the file has an "includes" to the `default.css` located in your controls directory).

#### Values:

`default`  
`none`  
`url`

##### b:htmleentities

If you set the `b:htmleentities="true"` attribute on the `body` tag, any doctype reference in a received XML document is resolved. If there is no doctype declared in the received XML file, a standard doctype is added that references a local file (the `entities.xml` file in the root of your Backbase installation directory.) This setting has a small performance penalty (it results in an extra request going to the server), therefore you should only use it when you use non-standard XML entities (the default is `false`).

##### b:debug

Set `b:debug="true"` in the `body` to log all the commands (`b:actions`), or on a tag or `b:behavior` just to log element or behavior specific actions. The logging statements are displayed in the Backbase Tools>Runtime Tracer.

### xhtml:form

Encloses a web form. A two tier submittal process is started when an input button with `type="submit"` is clicked or when a `submit` command is issued to the form. Within the `submit` event handler once all conditions set by the various behaviors have been satisfied then a `send` command should be issued. This takes care of the final submission. Using the `b:target` and `b:mode` attributes, you can load the submission *response file* anywhere

you want.

A form in a single page interface always requires a specified target.

#### Attributes

##### **b:destination**

Specify an XPath expression pointing to the receiving/related element. The **b:destination** of a `load`, `xsl-transform`, `send`, or `submit` command can be a variable, for example `b:destination="$myvar"`, in which case the variable is resolved to a String, or it can target an element in the BXML tree, in which case it is resolved to BXML.

##### **b:mode**

Define where a node is inserted relative to the receiving element. By default, the value is set to `replacechildren`.

Values:

- after
- asfirstchild
- aslastchild
- before
- replace
- replacechildren

## xhtml:link

The standard link tag with added functionality to make it easy to call different browser-dependent definitions.

#### Attributes

##### **b:browser**

A space separated list of values which correspond to client browsers. The tag that has this attribute is not processed unless the current browser is specified in the **b:browser** attribute value.

Values:

- ie
- ie5
- ie55
- ie6
- ie7
- moz
- quirk

## xhtml:meta

The standard `meta` tag is used to set initialization variables for the BPC. Any meta tag that has a name attribute with a value that starts with a '\$' will be picked up by the bpc and turned into a global variable for use in the SPI. This meta tag must be located in the `xhtml:head` element.

#### Implementation

```
<meta name="$bpc_controlpath" content="/Backbase/controls/basic/" />
```

## xhtml:script

Working with the `script` tag in a BXML (or XHTML) environment requires the script code to be properly **escaped** for the XML parser, because otherwise the <, > and & characters within your code will cause an XML parse error. If placing the script inside the body tag, the best method to escape code is to put the content of the `script` tag within CDATA sections.

You only need to escaping the script when it is used within the `xmp` tags (that mark the BXML space).

## Attributes

### b:browser

A space separated list of values which correspond to client browsers. The tag that has this attribute is not processed unless the current browser is specified in the `b:browser` attribute value.

Values:

ie  
ie5  
ie55  
ie6  
ie7  
moz  
quirk

## Implementation

```
<script>//<![CDATA[
    function show() {
        if (1< 2)
            alert("This message will not show without the CDATA section");
    }
    show();//]]>
</script>
```

## xhtml:style

The standard style tag with added functionality to make it easy to implement different browser-dependent style definitions.

## Attributes

### b:browser

A space separated list of values which correspond to client browsers. The tag that has this attribute is not processed unless the current browser is specified in the `b:browser` attribute value.

Values:

ie  
ie5  
ie55  
ie6  
ie7  
moz  
quirk

## xhtml:td

Extend the standard table cell with a `b:sortvalue` attribute to specify a value to sort on.

## Attributes

### b:sortvalue

Specify a string value that determines the cell is sorted by this value instead of by its contents.

## xhtml:xmp

Marks the start and end of the *BXML Space*. If the `b:backbase` attribute is set to `true` (the default), the content of the `xmp` tag is parsed by the BPC. An `xmp` tag that is processed by the BPC is converted to a `b:backbase` tag at runtime. You need to take this conversion into account for XPath statements that target `xmp` tags. You can have multiple

XMP tags in your body tag.

## Implementation

When a Backbase startup file is in the process of being rendered by the BPC, the content (i.e. the code) of the `xmp` tags is shown. The following example shows how to hide the content, but show the rendered page: you need to set `style="display:none;"` on the `xmp` tag and use a `construct` event, or an `s:execute`, to show the contents of the `xmp` tag. (The `s:execute` is executed when the entire page is rendered, therefore after the `construct` event.)

```
<xmp b:backbase="true" style="display:none;">
    <!--use s:execute or construct event to show contents-->
    <s:execute>
        <s:task b:action="show" />
    </s:execute>
    <!--s:event b:on="construct"> <s:task b:action="show" /> </s:event-->
    <!--your code here-->
</xmp>
```

However, using `style="display:none;"` can cause problems concerning the positioning of elements. For example, if you use the `maximize` command, when executed on `construct`, errors are thrown when an element is maximized before the display is no longer "none". In this case, use `visibility` instead of `display` to hide the content of the `xmp` tag:

```
<xmp b:backbase="true" style="visibility:hidden;">
    <s:execute>
        <s:task b:action="visible" />
    </s:execute>
    <!--your code here-->
</xmp>
```

## Attributes

### **b:backbase**

Enable or disable Backbase processing of the XMP tag.

## BXML Controls

### b:accordeon

The accordeon control groups related information together. It contains head and body elements that define each item in the group. The head element describes the content contained in the body and is always visible; you can expand or collapse the body elements by clicking the head element. Only one of the body elements can be selected at a time; expanding one item automatically collapses the currently selected item (using an animation effect). When selected, the body takes the size of the content.

#### Child Tags [Required]

b:accordeonbody , b:accordeonhead

#### Implementation

In the following example, the accordeon contains three accordeon head and body items. Note you can use the `b:state` attribute to define which of the heads opened by default.

```
<b:accordeon style="width: 500px">
  <b:accordeonhead>The Seven Samurai</b:accordeonhead>
  <b:accordeonbody>
    <p>A movie by Akira Kurosawa starring Takashi Shimura and Toshiro Mifune.</p>
  </b:accordeonbody>
  <b:accordeonhead>          2001: A Space Odyssey (1968)</b:accordeonhead>
  <b:accordeonbody>
    <p>An influential science fiction film directed by Stanley Kubrick.</p>
  </b:accordeonbody>
  <b:accordeonhead b:state="selected">      Life Is Beautiful (1997)</b:accordeonhead>
  <b:accordeonbody>
    <p>An Italian language film which tells the story of an Italian Jew, Guido Orefice (played by Roberto Benigni), who lives in a romantic fairy tale, but must learn how to use that dreamy quality to survive a concentration camp with his young son Joshua (played by Giorgio Cantarini).</p>
  </b:accordeonbody>
</b:accordeon>
```

### b:accordeonbody

The body element of an item in a `b:accordeon` in which you define the content that is displayed when its `b:accordeonhead` is selected.

#### Child Tags

You can define any BXML (and HTML) elements as children of this tag.

#### Parent Tags

b:accordeon

### b:accordeonhead

The head element of an item in a `b:accordeon`. Clicking the head displays the content defined in the `b:accordenbody`.

#### Child Tags

You can define any BXML (and HTML) elements as children of this tag.

#### Parent Tags

b:accordeon

### b:barchart

The barchart control consists of a series of labeled vertical bars that show different values for each bar - it is used to display graphically these differences. The barchart control consists of an axis and .

---

**Attributes****b:caption**

Specify the heading text for the chart.

**b:x-axis-label**

Specify the text for the x-axis.

**b:y-axis-label**

Specify the text for the y-axis.

**b:mode**

Specify whether the chart is displayed two or three dimensionally (3D is the default).

**Values:**

3d

normal

**[Required] b:name**

Specify a logical unique name with which to identify an element.

**[Required] b:width**

Specify a percentage or pixel value specifying the width of an element.

**[Required] b:height**

Specify a percentage or pixel value for the height of an element.

**Child Tags [Required]****b:barchart-horizontal-values , b:barchart-series**

---

**Implementation**

```
<b:barchart b:caption="backbase.com visitors from Russia from 9th of June 2005 till 13th of
June 2005" b:x-axis-label="Date" b:y-axis-label="Number of Visitors" b:name="name"
b:width="600" b:height="400">
  <b:barchart-horizontal-values>
    <b:barchart-value b:value="9 June 2005" />
    <b:barchart-value b:value="10 June 2005" />
    <b:barchart-value b:value="11 June 2005" />
    <b:barchart-value b:value="12 June 2005" />
    <b:barchart-value b:value="13 June 2005" />
  </b:barchart-horizontal-values>
  <b:barchart-series b:name="Visitors from Russia" b:color="ff0000">
    <b:barchart-bar b:value="15" />
    <b:barchart-bar b:value="23" />
    <b:barchart-bar b:value="25" />
    <b:barchart-bar b:value="22" />
    <b:barchart-bar b:value="26" />
  </b:barchart-series>
</b:barchart>
```

---

**b:barchart-bar**

A **b:barchart-bar** defines the size (the vertical dimension) of a bar in the barchart, specified using the **b:value** attribute. The highest value from all the **b:barchart-bars** is used to calculate ten equal steps that appear on the vertical axis.

---

**Attributes****b:value**

A string containing a stated value.

**Parent Tags [Required]**

---

**b:barchart-horizontal-values**

A container to define the horizontal axis of the barchart. Each **b:barchart-value** describes a bar in the barchart; the vertical axis of the bar is defined by a **b:barchart-bar** in the **b:barchart-series**.

---

**Child Tags [Required]****b:barchart-value****Parent Tags [Required]**

**b:barchart****b:barchart-series**

A container to define the vertical axis of the barchart. Each **b:barchart-bar** in the **b:barchart-series** is an integer value defining the vertical axis of the bar; on the horizontal axis, the **b:barchart-value** describes the bar.

**Attributes****b:name**

Specify a logical unique name with which to identify an element.

**Child Tags [Required]****b:barchart-bar****Parent Tags [Required]****b:barchart****b:barchart-value**

A **b:barchart-value** is a string text, defined using the **b:value** attribute, to describe a column in the chart.

**Attributes****b:value**

A string containing a stated value.

**Parent Tags [Required]****b:barchart-horizontal-values****b:box**

The box is a block level control that can contain any elements.

**Attributes****b:style**

Use the standard CCS properties and values to style the inner elements of a Backbase control, for example **b:style="width: 100px"**. Note that the standard HTML **style** attribute sets the style for the outer part of the control.

**Child Tags**

You can define any BXML (and HTML) elements as children of this tag.

**Implementation**

```
<b:box style="position: absolute;left: 300px; top:25px;" b:style="background-color: #CCFF00">
  "Scarface developed a cult following among younger audiences."</b:box>
```

**b:button**

A button control that allows the user to send a command, for example to submit a form.

**Attributes****b:innerstyle**

Use the standard CCS properties and values to style the content of a Backbase control, for example **b:innerstyle="background-color: black; color: white"**.

**b:style**

Use the standard CCS properties and values to style the inner elements of a Backbase control, for example **b:style="width: 100px"**. Note that the standard HTML **style** attribute sets the style for the outer part of the control.

**Child Tags**

You can define any BXML (and HTML) elements as children of this tag.

**Implementation**

In the following example, clicking the button shows an alert message.

```
<b:button b:action="alert" b:value="This button works!!!">      Styled button</b:button>
```

## b:combobox

Creates a textarea field (user-entry) with a drop-down listbox containing alternative values from a fixed set. The **b:combobox** contains child elements **b:combo-option** tags that define the list of options. The **b:combobox** tag **b:name** attribute is the name of the input field that is sent on a form submit. You can specify the initial value in the textarea field using the **b:value** attribute (a String value of a a combo-option), or **b:text** to set the initial text that appears in the box. The **b:combobox** also can have the **b:style** attribute to indicate the styling for the input field.

Due to a problem with Internet Explorer, you cannot use % to indicate width size (**b:width**) of a **b:combobox**.

### Attributes

#### [Required] **b:name**

Specify a logical unique name with which to identify an element.

#### **b:value**

A string containing a stated value.

#### **b:text**

Sets the text to display in the combobox

#### **b:width**

Specify a percentage or pixel value specifying the width of an element.

### Child Tags [Required]

#### **b:combo-option**

## Implementation

```
<b:combobox b:width="300px" b:name="myvalue" b:text="B">
  <b:combo-option b:value="1">2001: A Space Odyssey (1968), dir. Stanley Kubrick</b:combo-option>
  <b:combo-option b:value="2">A Clockwork Orange (1971), dir. Stanley Kubrick</b:combo-option>
  <b:combo-option b:value="3">A Hard Day's Night (1964), dir. Richard Lester</b:combo-option>
  <b:combo-option b:value="4">A Room With A View (1986), dir. James Ivory</b:combo-option>
  <b:combo-option b:value="5">A Streetcar Named Desire (1951), dir. Elia Kazan</b:combo-option>
  <b:combo-option b:value="6">Adam's Rib (1949), dir. George Cukor</b:combo-option>
  <b:combo-option b:value="7">All About Eve (1950), dir. Joseph Mankiewicz</b:combo-option>
  <b:combo-option b:value="8">All That Jazz (1979), dir. Bob Fosse</b:combo-option>
  <b:combo-option b:value="9">Annie Hall (1977), dir. Woody Allen</b:combo-option>
  <b:combo-option b:value="11">Annie Hall (1977), dir. Woody Allen</b:combo-option>
  <b:combo-option b:value="12">Apocalypse Now (1979), dir. Francis Ford Coppola</b:combo-option>
  <b:combo-option b:value="13">Blade Runner (1982), dir. Ridley Scott</b:combo-option>
  <b:combo-option b:value="14">Blue Velvet (1986), dir. David Lynch</b:combo-option>
  <b:combo-option b:value="15">Bonnie And Clyde (1967), dir. Arthur Penn</b:combo-option>
  <b:combo-option b:value="16">Bringing Up Baby (1938), dir. Howard Hawks</b:combo-option>
  <b:combo-option b:value="17">Casablanca (1942), dir. Michael Curtiz</b:combo-option>
  <b:combo-option b:value="18">Chinatown (1974), dir. Roman Polanski</b:combo-option>
  <b:combo-option b:value="19">Citizen Kane (1941), dir. Orson Welles</b:combo-option>
  <b:combo-option b:value="19">City Lights (1931), dir. Charles Chaplin</b:combo-option>
</b:combobox>
```

## b:combo-option

Defines an option in a list of options for a **b:combobox**. The **b:combo-option** tag **b:value** attribute is the value of a value-representation pair.

### Attributes

**b:value**

A string containing a stated value.

Parent Tags [Required]**b:combobox****b:contextmenu**

The Contextmenu is a context-sensitive menu that pops-up when an event, typically a right-mouse button event, is triggered on an element. You create the menu by nesting **b:contextmenu** tags, and you define each row in the menu using the **b:contextmenurow** tag.

Child Tags**[Required] b:contextmenurow**Implementation

In the following example, a right-mouse button (`rmbup`) event handler is defined. Right-click in the bordered element to trigger the event handler (that targets the context menu) and view the pop-up menu.

```
<div style="height: 500px; width: 600px; border: 1px solid green">
    <s:event b:on="rmbdown" b:bubble="false" />
    <s:event b:on="rmbup" b:bubble="false">
        <s:task b:action="select" b:target="id('contextmenu')" />
        <s:task b:action="position" b:type="place" b:target="id('contextmenu')"
b:destination=.." b:position="at-pointer" />
    </s:event>
    <b:contextmenu id="contextmenu">
        <b:contextmenurow b:label="Open" b:shortcut="ctrl + o" />
        <b:contextmenurow b:label="Close this file" />
        <b:contextmenurow b:label="Sub">
            <b:contextmenu>
                <b:contextmenurow b:label="Save" b:shortcut="ctrl+s" />
                <b:contextmenurow b:label="Something" />
                <b:contextmenurow b:label="Submenu">
                    <b:contextmenu>
                        <b:contextmenurow b:label="Something" />
                        <b:contextmenurow b:label="Something" b:shortcut="ctrl+a" />
                        <b:contextmenurow b:label="Something" />
                    </b:contextmenu>
                </b:contextmenurow>
                <b:contextmenurow b:label="Something" b:shortcut="ctrl+z" />
            </b:contextmenu>
        </b:contextmenurow>
        <b:contextmenurow b:label="Sub">
            <b:contextmenu>
                <b:contextmenurow b:label="Save" b:shortcut="ctrl+s" />
                <b:contextmenurow b:label="Something" />
                <b:contextmenurow b:label="Submenu" b:disabled="true">
                    <b:contextmenu>
                        <b:contextmenurow b:label="Something" />
                        <b:contextmenurow b:label="Something" b:shortcut="ctrl+a" />
                        <b:contextmenurow b:label="Something" />
                    </b:contextmenu>
                </b:contextmenurow>
                <b:contextmenurow b:label="Something" b:shortcut="ctrl+z" />
            </b:contextmenu>
        </b:contextmenurow>
        <b:contextmenurow b:label="Exit" b:shortcut="alt+F4" />
    </b:contextmenu>
</div>
<div b:dragreceive="true"></div>
```

**b:contextmenurow**

Defines a row in a **b:contextmenu** tag. The **b:contextmenurow** tag has a **b:label** attribute that defines the menu label, and optionally a **b:shortcut** attribute that adds a label

to denote a shortcut key.

#### Attributes

---

##### [Required] b:label

Specify a String value that displays text in a set location for a control.

##### b:shortcut

A String value that indicates a shortcut key for a menu item. The value is a text value that has no implementation. To implement the keys, use the s:keys tag.

---

##### Parent Tags [Required]

##### b:contextmenu

---

## b:datagrid

Displays cell-based tabular data and supports selecting, sorting, paging, and editing of the data, and has resizable rows and columns. The b:datagrid has child elements: b:datagridhead and b:datagridbody, and b:datatype.

The b:datagridhead defines the head and contains a b:datagridrow in which the header for each column is defined using a b:datagridgridheadcell element.

The b:datagridbody defines the body element. The content of the <b:datagridbody/> is an HTML table that must be dynamically loaded into the body using the b:url attribute. You can specify the following for the b:url attribute:

- An XML file that contains raw XML, that is transformed using one of the XSLTs defined in the attributes b:template or b:template-ie5
- An XML file that contains an HTML table body (attributes and children)
- Any other file type, such as PHP or ASP, that contains and returns raw XML. The XML is transformed using one of the XSLTs defined in the attributes b:template or b:template-ie5

The b:datagrid can also have b:datatype child elements that defines a form element that is applied in the cell of a table. Basically, you can define any HTML or BXML form element that fits within a cell, such as a select box, text input type, b:spinner, or b:datepicker

The control also has properties for defining a paging mechanism, in combination with the b:page control, the implementation of which you must define on the server-side.

The b:listgrid and b:datagrid controls trigger specific events that you can use to customize and extend the controls:

- **populate** - occurs when the page is constructed and is used to fill the data variable with an HTML tbody string, or an XML string (an XML string requires post processing using templates - for more information, see section **8.2 Browser Templatting** of the Manual PDF).

The controls/backbase/b-datagrid/b-datagrid.xml and controls/backbase/b-listgrid/b-listgrid.xml contain implementations of the populate

- **sort-column** - occurs when a column is sorted. It has the following variables available: sortReverse value: true | false and currentPosition value: column number.
- **selection-changed** - occurs when a selected column is changed. It has the following variables available: selected-rows and deselected-rows.
- **cell-changed** - occurs when a column is changed. It has the following variables available: row, cell, rowIndex, and cellIndex.
- **row-changed** - occurs when a row is changed. It has the following variables avail-

able: `row` and `rowIndex`.

When you use the `b:datagrid`, ensure that ancestor elements do not have the `style="display:none"` property set. (The control can experience problems when being drawn if this property is set.)

### Attributes

#### `b:url`

Specify the path of the file to load, relative to the startup file of your Backbase application.

#### `b:items-in-total`

The total number of records returned in the paged data set. The value is used to calculate the total number of pages, and should be placed on the `b:tilelist-body` every time the server serves it so that the tilelist knows the current size of the data set.

#### `b:page-size`

Define the number of records displayed in every page.

#### `b:page-number`

Define the initial page that is displayed when the page is constructed. When you navigate through the pages, the value is dynamically updated to indicate which page is currently displayed.

#### `b:page-cache`

Define the number of pages retained in memory (0 or 1 are identical).

#### `b:pages-in-total`

Read-only attribute that is calculated at runtime, if the control is connected to a paged data set, by dividing the `b:items-in-total` attribute by the `b:page-size` attribute.

#### `b:template`

Specify the name of template used if the browser offers full XSLT 1.0 support.

#### `b:ie5-template`

Specify the name of template used if the browser does not offer full XSLT 1.0 support.

### Child Tags [Required]

`b:datagridbody` , `b:datagridhead` , `b:datatype`s

## Implementation

The code snippets below provide examples on using the datagrid where the `b:url` points to raw XML, in an XML file or PHP file which is then converted using XSLTs, or when it points to a file in which the data is already formatted as an HTML table.

In the following example, the `b:url` specifies an XML file containing raw XML, and uses the XSLTs in the `b:template` and `b:template-ie5` to transform the data:

```
<b:datagrid style="width:800px; height:350px;" b:url="datagrid_film.xml"
b:template="xsl/b:datagrid.xsl" b:ie5-template="xsl/b:datagrid-ie5.xsl">
  <b:datagridhead>
    <b:datagridrow>
      <b:datagridheadcell />
      <b:datagridheadcell>Title</b:datagridheadcell>
      <b:datagridheadcell>Director</b:datagridheadcell>
      <b:datagridheadcell>Genre</b:datagridheadcell>
      <b:datagridheadcell>Language</b:datagridheadcell>
      <b:datagridheadcell>Premiere</b:datagridheadcell>
    </b:datagridrow>
  </b:datagridhead>
  <b:datagridbody />
</b:datagrid>
```

Where the XML data for `b:url` is as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<films>
  <film id="film-1">
```

```

<title>The Seven Samurai</title>
<director>Akira Kurosawa</director>
<genre>Drama</genre>
<language>Japanese</language>
<year>1954-01-12</year>
</film>
<film id="film-2">
    <title>City of God</title>
    <director>Fernando Meirelles</director>
    <genre>Drama</genre>
    <language>Portuguese</language>
    <year>2002-02-03</year>
</film>
<film id="film-3">
    <title>The Godfather</title>
    <director>Francis Ford Coppola</director>
    <genre>Crime</genre>
    <language>English</language>
    <year>1972-09-17</year>
</film>
</films>

```

Where the XSLT used for **b:template** is as follows:

```

<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" b:name="stylesheet">
    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"
    omit-xml-declaration="yes" />
    <xsl:template match="/">
        <tbody>
            <xsl:apply-templates select="*/*[position()]"/>
                <xsl:sort select="text" order="ascending" data-type="number" />
            </xsl:apply-templates>
        </tbody>
    </xsl:template>
    <xsl:template match="*">
        <tr class="b-datagrid-tr">
            <th onmouseover="__addClass('b-datagridrowhead-hov', this);"
            onmouseout="__removeClass('b-datagridrowhead-hov', this); class='b-datagridrowhead">
                <div class="b-datagridrowhead-div">
                    <xsl:value-of select="position()" />
                </div>
            </th>
            <xsl:for-each select="*">
                <td onmouseover="__addClass('b-datagridcell-hov', this);"
                onmouseout="__removeClass('b-datagridcell-hov', this); class='b-datagridcell">
                    <div class="b-datagridcell-div">
                        <xsl:value-of select="." />
                    </div>
                </td>
            </xsl:for-each>
        </tr>
    </xsl:template>
</xsl:stylesheet>

```

Where the XSLT used for **b:template-ie5** is as follows:

```

<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0">
    <xsl:template match="/">
        <tbody>
            <xsl:apply-templates select="*/*" order-by="+ number(text)" />
        </tbody>
    </xsl:template>
    <xsl:template match="*">
        <tr class="b-datagrid-tr">
            <th class="b-datagridrowhead">
                <div class="b-datagridrowhead-div">x</div>
            </th>
            <xsl:for-each select="*">
                <td class="b-datagridcell">
                    <div class="b-datagridcell-div">
                        <xsl:value-of select="." />
                    </div>
                </td>
            </xsl:for-each>
        </tr>
    </xsl:template>
</xsl:stylesheet>

```

```

        </xsl:for-each>
    </tr>
</xsl:template>
</xsl:stylesheet>

```

In the following example, the **b:url** specifies an XML file containing preformed data:

```

<b:datagrid style="width:800px; height:350px;" b:url="preformed_datagrid_film.xml">
    <b:datagridhead>
        <b:datagridrow>
            <b:datagridheadcell />
            <b:datagridheadcell>Title</b:datagridheadcell>
            <b:datagridheadcell>Director</b:datagridheadcell>
            <b:datagridheadcell>Genre</b:datagridheadcell>
            <b:datagridheadcell>Language</b:datagridheadcell>
            <b:datagridheadcell>Premiere</b:datagridheadcell>
        </b:datagridrow>
    </b:datagridhead>
    <b:datagridbody />
</b:datagrid>

```

Where the XML data for **b:url** is as follows:

```

<?xml version="1.0" encoding="UTF-8" ?>
<tbody xmlns="http://www.w3.org/1999/xhtml" xmlns:b="http://www.backbase.com/b"
xmlns:s="http://www.backbase.com/s">
<tr class="b-datagrid-tr">
    <th class="b-datagridrowhead" onmouseout="__removeClass('b-datagridrowhead-hov', this);"
onmouseover="__addClass('b-datagridrowhead-hov', this);">
        <div class="b-datagridrowhead-div">1</div>
    </th>
    <td class="b-datagridcell" onmouseout="__removeClass('b-datagridcell-hov', this);"
onmouseover="__addClass('b-datagridcell-hov', this);">
        <div class="b-datagridcell-div">The Seven Samurai</div>
    </td>
    <td class="b-datagridcell" onmouseout="__removeClass('b-datagridcell-hov', this);"
onmouseover="__addClass('b-datagridcell-hov', this);">
        <div class="b-datagridcell-div">Akira Kurosawa</div>
    </td>
    <td class="b-datagridcell" onmouseout="__removeClass('b-datagridcell-hov', this);"
onmouseover="__addClass('b-datagridcell-hov', this);">
        <div class="b-datagridcell-div">Drama</div>
    </td>
    <td class="b-datagridcell" onmouseout="__removeClass('b-datagridcell-hov', this);"
onmouseover="__addClass('b-datagridcell-hov', this);">
        <div class="b-datagridcell-div">Japanese</div>
    </td>
    <td class="b-datagridcell" onmouseout="__removeClass('b-datagridcell-hov', this);"
onmouseover="__addClass('b-datagridcell-hov', this);">
        <div class="b-datagridcell-div">1954-01-12</div>
    </td>
</tr>
<tr class="b-datagrid-tr">
    <th class="b-datagridrowhead" onmouseout="__removeClass('b-datagridrowhead-hov', this);"
onmouseover="__addClass('b-datagridrowhead-hov', this);">
        <div class="b-datagridrowhead-div">2</div>
    </th>
    <td class="b-datagridcell" onmouseout="__removeClass('b-datagridcell-hov', this);"
onmouseover="__addClass('b-datagridcell-hov', this);">
        <div class="b-datagridcell-div">City of God</div>
    </td>
    <td class="b-datagridcell" onmouseout="__removeClass('b-datagridcell-hov', this);"
onmouseover="__addClass('b-datagridcell-hov', this);">
        <div class="b-datagridcell-div">Fernando Meirelles</div>
    </td>
    <td class="b-datagridcell" onmouseout="__removeClass('b-datagridcell-hov', this);"
onmouseover="__addClass('b-datagridcell-hov', this);">
        <div class="b-datagridcell-div">Drama</div>
    </td>
    <td class="b-datagridcell" onmouseout="__removeClass('b-datagridcell-hov', this);"
onmouseover="__addClass('b-datagridcell-hov', this);">
        <div class="b-datagridcell-div">Portuguese</div>
    </td>
</tr>

```

```

        <div class="b-datagridcell-div">2002-02-03</div>
    </td>
</tr>
<tr class="b-datagrid-tr">
    <th class="b-datagridrowhead" onmouseout="__removeClass('b-datagridrowhead-hov', this);"
onmouseover="__addClass('b-datagridrowhead-hov', this);">
        <div class="b-datagridrowhead-div">3</div>
    </th>
    <td class="b-datagridcell" onmouseout="__removeClass('b-datagridcell-hov', this);"
onmouseover="__addClass('b-datagridcell-hov', this);">
        <div class="b-datagridcell-div">The Godfather</div>
    </td>
    <td class="b-datagridcell" onmouseout="__removeClass('b-datagridcell-hov', this);"
onmouseover="__addClass('b-datagridcell-hov', this);">
        <div class="b-datagridcell-div">Francis Ford Coppola</div>
    </td>
    <td class="b-datagridcell" onmouseout="__removeClass('b-datagridcell-hov', this);"
onmouseover="__addClass('b-datagridcell-hov', this);">
        <div class="b-datagridcell-div">Crime</div>
    </td>
    <td class="b-datagridcell" onmouseout="__removeClass('b-datagridcell-hov', this);"
onmouseover="__addClass('b-datagridcell-hov', this);">
        <div class="b-datagridcell-div">English</div>
    </td>
    <td class="b-datagridcell" onmouseout="__removeClass('b-datagridcell-hov', this);"
onmouseover="__addClass('b-datagridcell-hov', this);">
        <div class="b-datagridcell-div">1972-09-17</div>
    </td>
</tr>
</tbody>
```

In the following example, the **b:url** specifies a PHP file containing raw XML, and uses the XSLTs in the **b:template** and **b:template-ie5** to transform the data:

```

<b:datagrid style="width:800px; height:350px;" b:url="films_raw.php"
b:template="xsl/b-datagrid.xsl" b:ie5-template="xsl/b-datagrid-ie5.xsl">
    <b:datagridhead>
        <b:datagridrow>
            <b:datagridheadcell />
            <b:datagridheadcell>Title</b:datagridheadcell>
            <b:datagridheadcell>Director</b:datagridheadcell>
            <b:datagridheadcell>Genre</b:datagridheadcell>
            <b:datagridheadcell>Language</b:datagridheadcell>
            <b:datagridheadcell>Premiere</b:datagridheadcell>
        </b:datagridrow>
    </b:datagridhead>
    <b:datagridbody />
</b:datagrid>
```

Where the PHP file for **b:url** is as follows (PHP must be installed on your web server):

```

<?php header('Content-Type: text/xml'); echo '<?xml version="1.0" encoding="UTF-8"?>
'; ?>
<films>
    <film id="film-1">
        <title>The Seven Samurai</title>
        <director>Akira Kurosawa</director>
        <genre>Drama</genre>
        <language>Japanese</language>
        <year>1954-01-12</year>
    </film>
    <film id="film-2">
        <title>City of God</title>
        <director>Fernando Meirelles</director>
        <genre>Drama</genre>
        <language>Portuguese</language>
        <year>2002-02-03</year>
    </film>
    <film id="film-3">
        <title>The Godfather</title>
        <director>Francis Ford Coppola</director>
        <genre>Crime</genre>
        <language>English</language>
        <year>1972-09-17</year>
    </film>
</films>
```

```
</film>
</films>
```

## b:datagridbody

Defines the body section of a datagrid, into which the content (an HTML table) is loaded. Specify the data using the [b:url](#) attribute.

[Parent Tags \[Required\]](#)

[b:datagrid](#)

## b:datagridhead

Defines the head section of a datagrid, defined using the [b:datagridrow](#) tag.

[Child Tags \[Required\]](#)

[b:datagridrow](#)

[Parent Tags \[Required\]](#)

[b:datagrid](#)

## b:datagridheadcell

Defines a cell in a [b:datagridhead](#). The [b:datatype](#) attribute specifies the name of the datatype used in the cell.

[Attributes](#)

[\[Required\] b:datatype](#)

Specify the datatype used in a cell of a grid control. The value is the [b:name](#) of a [b:datatype](#) element declared within the listgrid or datagrid.

[Parent Tags \[Required\]](#)

[b:datagridhead](#)

## b:datagridrow

Defines a row in the body or head section of a datagrid.

[Parent Tags \[Required\]](#)

[b:datagridbody , b:datagridhead](#)

[Child Tags \[Required\]](#)

[b:datagridheadcell](#)

## b:datatype

Defines a datatype that you can use in a datagrid or listgrid. The allowed types are any HTML or BXML form element, such as a selectbox, text input type, spinner, or datepicker, basically any element that fits within a cell.

[Attributes](#)

[\[Required\] b:name](#)

Specify a logical unique name with which to identify an element.

[Parent Tags \[Required\]](#)

[b:datatype](#)

## b:datatypes

Container for defining the allowed types in a datagrid or listgrid.

[Parent Tags \[Required\]](#)

[b:datagrid](#)

[Child Tags \[Required\]](#)

**b:datatype****Implementation**

```
<b:datatypes>
  <b:datatype b:name="text">
    <input type="text" value="default" b:focusitem="true" b:focusgroup="true" />
  </b:datatype>
  <b:datatype b:name="numbers">
    <select>
      <option>1</option>
      <option>2</option>
      <option>3</option>
      <option>4</option>
      <option>5</option>
    </select>
  </b:datatype>
  <b:datatype b:name="letters">
    <select>
      <option>a</option>
      <option>b</option>
      <option>c</option>
      <option>d</option>
      <option>e</option>
    </select>
  </b:datatype>
  <b:datatype b:name="spinner">
    <b:spinner />
  </b:datatype>
</b:datatypes>
```

**b:datepicker**

Allows users to select a date from a calendar that is then displayed in an input element. You can specify the date format displayed in the input element using the **b:format** attribute, and an initial value using **b:value** attribute using a date that conforms to the date format in the **b:format** attribute.

**Attributes****b:type**

Specify where the datepicker appears relative to its input type. Set **float** (the default) to display a clickable calendar icon next to the input element (clicking the icon opens the calendar), or **inline** to display the calendar underneath the input element.

Values:  
**float**  
**inline**

**b:input**

Specify an XPath statement to the input element that is used to display the selected date.

**b:format**

Specify the following date formats:

- **d** - displays the day as a number without a leading zero (for example, 1)
- **dd** - displays the day as a number with a leading zero (for example, 01)
- **ddd** - displays the day as an abbreviation (for example, Sun)
- **ddd** - displays the day as a full name (for example, Sunday)
- **M** - displays the month as a number without a leading zero (for example, January is represented as 1)
- **MM** - displays the month as a number with a leading zero (for example, 01/12/01)
- **MMM** - displays the month as an abbreviation (for example, Jan)
- **MMMM** - displays the month as a full month name (for example, January)
- **y** - displays the year number (0-9) without leading zeros
- **yy** - displays the year in two-digit numeric format with a leading zero, if applicable

- `yyy` - displays the year in three digit numeric format
  - `yyyy` - Displays the year in four digit numeric format
- For example, `M/d/yy` is 12/7/58, `d-MMM` is 7-Dec, `d-MMMM-yy` is 7-December-58, `MMMM` is 7 December, and `MMMM yy` is December 58.

**b:value**

A string containing a stated value.

**Implementation**

```
<div>
  <h2>Datepicker Control (inline)</h2>
  <input type="text" name="input1" />
  <b:datepicker b:type="inline" b:input=".../input[1]" b:format="dddd dd MMMM yyyy" />
</div>
<div style="position: absolute; left: 300px;">
  <h2>Datepicker Control (float)</h2>
  <input type="text" name="input1" />
  <b:datepicker b:input=".../input[1]" b:format="dddd dd MMMM yyyy" />
</div>
```

**b:detailviewer**

Presents a detailed view of data such as those used to present technical product details. It presents the data in a table consisting of two columns: a description column and a detail column. Each row of the table is defined using a child `b:property` tag that has a `b:label` attribute to define the text value of the left-hand side column (description).

**Child Tags [Required]**

[Required] `b:property`

**Implementation**

```
<b:detailviewer>
  <b:property b:label="Movie" b:style="width: 400px">Finding Nemo</b:property>
  <b:property b:label="Directors">Andrew Stanton
    <br />
  Lee Unkrich (co-director)
  </b:property>
  <b:property b:label="Genre">Animation / Comedy / Family</b:property>
  <b:property b:label="Plot Outline">A father/son underwater adventure featuring Nemo, a boy clownfish, stolen from his coral reef home. His timid father must then search the ocean to find him.</b:property>
</b:detailviewer>
```

**b:flash**

Renders the flash movie specified using the `b:url` attribute. You can set the width and the height using the `b:width` and `b:height` attributes. The `b:url` attribute cannot be dynamically constructed or updated, it has to be a static value.

**Attributes**

[Required] `b:url`

Specify the path of the file to load, relative to the startup file of your Backbase application.

`b:width`

Specify a percentage or pixel value specifying the width of an element.

`b:height`

Specify a percentage or pixel value for the height of an element.

**Implementation**

The following is an example of a flash control:

```
<b:flash b:url="/Backbase/controls/backbase/b-flash/tags.swf" b:width="300" b:height="300" />
```

## b:linechart

A line graph Used to show continuing data and to clearly present the progress and decline (for example, the peaks and troughs of stock price) of whatever the linechart represents.

### Attributes

#### b:caption

Specify the heading text for the chart.

#### b:x-axis-label

Specify the text for the x-axis.

#### b:y-axis-label

Specify whether the chart is displayed two or three dimensionally (3D is the default).

##### Values:

3d

normal

#### [Required] b:name

Specify a logical unique name with which to identify an element.

#### [Required] b:width

Specify a percentage or pixel value specifying the width of an element.

#### [Required] b:height

Specify a percentage or pixel value for the height of an element.

### Child Tags [Required]

#### b:linechart-horizontal-values , b:linechart-series

## Implementation

```
<b:linechart b:caption="backbase.com visitors from Russia, Bulgaria and England from 9th of June 2005 till 13th of June 2005" b:x-axis-label="Date" b:y-axis-label="Number of Visitors" b:width="600" b:height="400" b:name="name">
  <b:linechart-horizontal-values>
    <b:linechart-value b:value="9 June 2005" />
    <b:linechart-value b:value="10 June 2005" />
    <b:linechart-value b:value="11 June 2005" />
    <b:linechart-value b:value="12 June 2005" />
    <b:linechart-value b:value="13 June 2005" />
  </b:linechart-horizontal-values>
  <b:linechart-series b:name="Visitors from Russia" b:color="#990000">
    <b:linechart-point b:value="15" />
    <b:linechart-point b:value="23" />
    <b:linechart-point b:value="25" />
    <b:linechart-point b:value="22" />
    <b:linechart-point b:value="26" />
  </b:linechart-series>
  <b:linechart-series b:name="Visitors from Bulgaria" b:color="#000099">
    <b:linechart-point b:value="13" />
    <b:linechart-point b:value="14" />
    <b:linechart-point b:value="12" />
    <b:linechart-point b:value="13" />
    <b:linechart-point b:value="15" />
  </b:linechart-series>
  <b:linechart-series b:name="Visitors from England" b:color="#009900">
    <b:linechart-point b:value="15" />
    <b:linechart-point b:value="20" />
    <b:linechart-point b:value="27" />
    <b:linechart-point b:value="25" />
    <b:linechart-point b:value="25" />
  </b:linechart-series>
</b:linechart>
```

## b:linechart-horizontal-values

A container to define the values, **b:linechart-value**, for the horizontal axis of the chart. Each **b:linechart-value** is a string text to describe a point in the chart. The point size is determined by a **b:linechart-point** in the **b:linechart-series**.

Child Tags [Required]**b:linechart-value**Parent Tags [Required]**b:linechart**

## b:linechart-point

A **b:linechart-bar** is an integer value, defined using the **b:value** attribute, to denote the size (the vertical dimension) of a point in the chart. The highest value from all the **b:linechart-points** is used to calculate ten equal steps that appear on the vertical axis.

Parent Tags [Required]

## b:linechart-series

A container to define the values, **b:linechart-point**, for the vertical axis of the chart.

Each **b:linechart-point** is an integer value to denote the size of a point in the chart.

Attributes**b:name**

Specify a logical unique name with which to identify an element.

**b:color**

Describe the foreground color of an element's text content.

Values:

<color>

inherit

Child Tags [Required]**b:linechart-point**Parent Tags [Required]**b:linechart**

## b:linechart-value

A **b:linechart-value** is a string text, defined using the **b:value** attribute, to describe a point in the chart.

Attributes**b:value**

A string containing a stated value.

Parent Tags [Required]**b:linechart-horizontal-values**

## b:listgrid

Displays row-based tabular data, and supports selecting, paging, and editing of the data, and has resizable rows and columns. The **b:listgrid** has child elements: **b:listgridhead** and **b:listgridbody**, and have **b:datatype**s.

The **b:listgridhead** defines the head and contains a **b:listgridrow** in which the header for each column is defined using a **b:listgridheadcell** element.

The **b:listgridbody** defines the body element. The content of the **<b:listgridbody>** is an HTML table that must be dynamically loaded into the body using the **b:url** attribute. You can specify the following for the **b:url** attribute:

- An XML file that contains raw XML, that is transformed using one of the XSLTs defined in the attributes **b:template** or **b:template-ie5**
- An XML file that contains an HTML table body (attributes and children)
- Any other file type, such as PHP or ASP, that contains and returns raw XML. The

XML is transformed using one of the XSLTs defined in the attributes `b:template` or `b:template-ie5`

The `b:listgrid` can also have `b:datatype` child elements that defines a form element that is applied in the cell of a table. Basically, you can define any HTML or BXML form element that fits within a cell, such as a select box, text input type, `b:spinner`, or `b:datepicker`

The control also has properties for defining a paging mechanism, in combination with the `b:page` control, the implementation of which you must define on the server-side.

The `b:listgrid` and `b:datagrid` controls trigger specific events that you can use to customize and extend the controls:

- `populate` - occurs when the page is constructed and is used to fill the `data` variable with an HTML table string, or an XML string (an XML string requires post processing using templates - for more information, see section **8.2 Browser Templatting** of the Manual PDF).

The `controls/backbase/b-datasource/b-datasource.xml` and `controls/backbase/b-listgrid/b-listgrid.xml` contain implementations of the `populate`

- `selection-changed` - occurs when a selected column is changed. It has the following variables available: `selected-rows` and `deselected-rows`.
- `cell-changed` - occurs when a cell is changed. It has the following variables available: `row`, `cell`, `rowIndex`, and `cellIndex`.
- `row-changed` - occurs when a row is changed. It has the following variables available: `row` and `rowIndex`.

When you use the `b:listgrid`, ensure that ancestor elements do not have the `style="display:none"` property set. (The control can experience problems when being drawn if this property is set.)

## Attributes

---

### `b:url`

Specify the path of the file to load, relative to the startup file of your Backbase application.

### `b:items-in-total`

The total number of records returned in the paged data set. The value is used to calculate the total number of pages, and should be placed on the `b:tilelist-body` every time the server serves it so that the tilelist knows the current size of the data set.

### `b:page-size`

Define the number of records displayed in every page.

### `b:page-number`

Define the initial page that is displayed when the page is constructed. When you navigate through the pages, the value is dynamically updated to indicate which page is currently displayed.

### `b:page-cache`

Define the number of pages retained in memory (0 or 1 are identical).

### `b:pages-in-total`

Read-only attribute that is calculated at runtime, if the control is connected to a paged data set, by dividing the `b:items-in-total` attribute by the `b:page-size` attribute.

### `b:template`

Specify the name of template used if the browser offers full XSLT 1.0 support.

### `b:ie5-template`

Specify the name of template used if the browser does not offer full XSLT 1.0 support.

Child Tags [Required]**b:datatype** , **b:listgridbody** , **b:listgridhead**Implementation

The code snippets below provide examples on using the listgrid where the **b:url** points to raw XML, in an XML file or PHP file which is then converted using XSLTs, or when it points to a file in which the data is already formatted as an HTML table.

In the following example, the **b:url** specifies an XML file containing raw XML, and uses the XSLTs in the **b:template** and **b:template-ie5** to transform the data:

```
<b:listgrid style="width:800px; height:350px;" b:url="films_raw.xml"
b:template="b-listgrid.xsl" b:ie5-template="b-listgrid-ie5.xsl">
    <b:listgridhead>
        <b:listgridrow>
            <b:listgridheadcell>Title</b:listgridheadcell>
            <b:listgridheadcell>Director</b:listgridheadcell>
            <b:listgridheadcell>Genre</b:listgridheadcell>
            <b:listgridheadcell>Language</b:listgridheadcell>
            <b:listgridheadcell>Premiere</b:listgridheadcell>
        </b:listgridrow>
    </b:listgridhead>
    <b:listgridbody />
</b:listgrid>
```

Where the XML data for **b:url** is as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<films>
    <film id="film-1">
        <title>The Seven Samurai</title>
        <director>Akira Kurosawa</director>
        <genre>Drama</genre>
        <language>Japanese</language>
        <year>1954-01-12</year>
    </film>
    <film id="film-2">
        <title>City of God</title>
        <director>Fernando Meirelles</director>
        <genre>Drama</genre>
        <language>Portuguese</language>
        <year>2002-02-03</year>
    </film>
    <film id="film-3">
        <title>The Godfather</title>
        <director>Francis Ford Coppola</director>
        <genre>Crime</genre>
        <language>English</language>
        <year>1972-09-17</year>
    </film>
</films>
```

Where the XSLT used for **b:template** is as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" b:name="stylesheet">
    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"
omit-xml-declaration="yes" />
    <xsl:template match="/">
        <tbody>
            <xsl:apply-templates select="*/*">
                <xsl:sort select="spinner" order="ascending" data-type="number" />
            </xsl:apply-templates>
        </tbody>
    </xsl:template>
    <xsl:template match="*">
        <tr class="b-listgridrow" onmouseover="bpc.addClass('b-listgridrow-hov', this);"
onmouseout="bpc.removeClass('b-listgridrow-hov', this);">
            <xsl:for-each select="*">
                <td class="b-listgridcell">
                    <div class="b-listgridcell-div">
                        <xsl:value-of select=".." />
                    </div>
                </td>
            </xsl:for-each>
        </tr>
    </xsl:template>
</xsl:stylesheet>
```

```

        </div>
    </td>
</xsl:for-each>
</tr>
</xsl:template>
</xsl:stylesheet>
```

Where the XSLT used for **b:template-ie5** is as follows:

```

<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0">
    <xsl:template match="/">
        <tbody>
            <xsl:apply-templates select="*/*" order-by="+ number(spinner)" />
        </tbody>
    </xsl:template>
    <xsl:template match="*">
        <tr class="b-listgridrow" onmouseover="bpc.addClass('b-listgridrow-hov', this);"
onmouseout="bpc.removeClass('b-listgridrow-hov', this);">
            <xsl:for-each select="*">
                <td class="b-listgridcell">
                    <div class="b-listgridcell-div">
                        <xsl:value-of select="." />
                    </div>
                </td>
            </xsl:for-each>
        </tr>
    </xsl:template>
</xsl:stylesheet>
```

In the following example, the **b:url** specifies an XML file containing preformed data:

```

<b:listgrid style="width:800px; height:350px;" b:url="preformedfilms_raw.xml">
    <b:listgridhead>
        <b:listgridrow>
            <b:listgridheadcell>Title</b:listgridheadcell>
            <b:listgridheadcell>Director</b:listgridheadcell>
            <b:listgridheadcell>Genre</b:listgridheadcell>
            <b:listgridheadcell>Language</b:listgridheadcell>
            <b:listgridheadcell>Premiere</b:listgridheadcell>
        </b:listgridrow>
    </b:listgridhead>
    <b:listgridbody />
</b:listgrid>
```

Where the XML data for **b:url** is as follows:

```

<?xml version="1.0" encoding="UTF-8" ?>
<tbody xmlns="http://www.w3.org/1999/xhtml" xmlns:b="http://www.backbase.com/b"
xmlns:s="http://www.backbase.com/s">
<tr onmouseout="bpc.removeClass('b-listgridrow-hov', this);"
onmouseover="bpc.addClass('b-listgridrow-hov', this); class='b-listgridrow'">
    <td class="b-listgridcell">
        <div class="b-listgridcell-div">The Seven Samurai</div>
    </td>
    <td class="b-listgridcell">
        <div class="b-listgridcell-div">Akira Kurosawa</div>
    </td>
    <td class="b-listgridcell">
        <div class="b-listgridcell-div">Drama</div>
    </td>
    <td class="b-listgridcell">
        <div class="b-listgridcell-div">Japanese</div>
    </td>
    <td class="b-listgridcell">
        <div class="b-listgridcell-div">1954-01-12</div>
    </td>
</tr>
<tr onmouseout="bpc.removeClass('b-listgridrow-hov', this);"
onmouseover="bpc.addClass('b-listgridrow-hov', this); class='b-listgridrow'">
    <td class="b-listgridcell">
        <div class="b-listgridcell-div">City of God</div>
    </td>
    <td class="b-listgridcell">
        <div class="b-listgridcell-div">Fernando Meirelles</div>
    </td>
```

```

        </td>
        <td class="b-listgridcell">
            <div class="b-listgridcell-div">Drama</div>
        </td>
        <td class="b-listgridcell">
            <div class="b-listgridcell-div">Portuguese</div>
        </td>
        <td class="b-listgridcell">
            <div class="b-listgridcell-div">2002-02-03</div>
        </td>
    </tr>
    <tr onmouseout="bpc.removeClass('b-listgridrow-hov', this);"
       onmouseover="bpc.addClass('b-listgridrow-hov', this); class='b-listgridrow'>
        <td class="b-listgridcell">
            <div class="b-listgridcell-div">The Godfather</div>
        </td>
        <td class="b-listgridcell">
            <div class="b-listgridcell-div">Francis Ford Coppola</div>
        </td>
        <td class="b-listgridcell">
            <div class="b-listgridcell-div">Crime</div>
        </td>
        <td class="b-listgridcell">
            <div class="b-listgridcell-div">English</div>
        </td>
        <td class="b-listgridcell">
            <div class="b-listgridcell-div">1972-09-17</div>
        </td>
    </tr>
</tbody>

```

In the following example, the **b:url** specifies a PHP file containing raw XML, and uses the XSLTs in the **b:template** and **b:template-ie5** to transform the data:

```

<b:listgrid style="width:800px; height:350px;" b:url="films_raw.php"
b:template="xsl/b-listgrid.xsl" b:ie5-template="xsl/b-listgrid-ie5.xsl">
    <b:listgridhead>
        <b:listgridrow>
            <b:listgridheadcell>Title</b:listgridheadcell>
            <b:listgridheadcell>Director</b:listgridheadcell>
            <b:listgridheadcell>Genre</b:listgridheadcell>
            <b:listgridheadcell>Language</b:listgridheadcell>
            <b:listgridheadcell>Premiere</b:listgridheadcell>
        </b:listgridrow>
    </b:listgridhead>
    <b:listgridbody />
</b:listgrid>

```

Where the PHP file for **b:url** is as follows (PHP must be installed on your web server):

```

<?php header('Content-Type: text/xml'); echo '<?xml version="1.0" encoding="UTF-8"?>
'; ?>
<films>
    <film id="film-1">
        <title>The Seven Samurai</title>
        <director>Akira Kurosawa</director>
        <genre>Drama</genre>
        <language>Japanese</language>
        <year>1954-01-12</year>
    </film>
    <film id="film-2">
        <title>City of God</title>
        <director>Fernando Meirelles</director>
        <genre>Drama</genre>
        <language>Portuguese</language>
        <year>2002-02-03</year>
    </film>
    <film id="film-3">
        <title>The Godfather</title>
        <director>Francis Ford Coppola</director>
        <genre>Crime</genre>
        <language>English</language>
        <year>1972-09-17</year>
    </film>
</films>

```

```
</films>
```

## b:listgridbody

Defines the body section of a listgrid, into which the content (an HTML table) is loaded. Specify the data using the `b:url` attribute.

Parent Tags [Required]

`b:listgrid`

## b:listgridhead

Defines the head section of a listgrid, defined using the `b:listgridrow` tag.

Child Tags [Required]

`b:listgridrow`

Parent Tags [Required]

`b:listgrid`

## b:listgridheadcell

Defines a cell in a `b:listgridhead`. The `b:datatype` attribute specifies the name of the datatype used in the cell.

Attributes

[Required] b:datatype

Specify the datatype used in a cell of a grid control. The value is the `b:name` of a `b:datatype` element declared within the listgrid or datagrid.

Parent Tags [Required]

`b:listgridhead`

## b:listgridrow

Defines a row in the body or head section of a listgrid.

Parent Tags [Required]

`b:listgridbody` , `b:listgridhead`

Child Tags [Required]

`b:listgridheadcell`

## b:livegrid

Displays data in a grid that user can scrolls through to load data records dynamically.

The `b:datasource` is required and can contain a query string. The start page and record page is always appended automatically. For example:

- `data.php?action=load` results in `data.php?action=load&start=1&length=10`
- `data.php` results in `data.php?start=1&length=10`

If you do not specify a `b:stylesheet`, the `stylesheet.xsl` file in the `/b-livegrid` directory is used. The result set must return the total number of results, which must be specified on the root tag of the data by an attribute. Specify the name of the attribute using the `b:total-attribute`.

It is important that the rows that are created by the stylesheet for the loaded data are exactly the same height as the prerendered (empty) rows. For example, the height of an empty row in the livegrid is by default set to 50px in the `b:livegrid.xml` file.

```
.b-livegrid td { height:50px; border-bottom: 1px solid green; }
```

The demo server file for the livegrid control only works if you have PHP 5 installed on

your server.

#### Attributes

##### [Required] b:datasource

Specify the XML datasource.

When the XML is part of the main BXML body, wrapped in an `s:xml` tag, the `b:name` of the datasource is stored in a variable that you can access using the syntax `$name`.

If you use Browser templating (`xsl-transform` command), the `b:datasource` attribute must refer to variable that contains XML.

##### [Required] b:stylesheet

Invoke a template stylesheet that defines the starting point of a block of processing/transformation rules.

If you use Backbase templating (`transform` command), the `b:name` attribute of the `s:stylesheet` tag is stored in a variable that you can access using the syntax `$name`.

If you use Browser templating (`xsl-transform` command), the `b:stylesheet` attribute must refer to variable that contains XML.

##### b:total-attribute

Specify the attribute name that implements the number of records that are returned from the data set.

##### b:load-at-once

Specify the number of records that are loaded each time the user scrolls the `b:livegrid`. By default, 10 records are loaded at a time.

#### Implementation

The following example is from the `example.xml` file located in the `Backbase/controls/backbase/b-livegrid` directory:

```
<div>
  <b:livegrid-listener b:listento=".../~*[1]" style="padding:5px;
position:absolute;right:0px;" />
</div>
<b:livegrid b:datasource="datasource.php" b:stylesheet="stylesheet.xsl"
b:total-attribute="total" b:load-at-once="24" />
```

## b:livegrid-listener

Displays paging information for a `b:livegrid` control.

#### Attributes

##### b:listento

Specify an XPath expression targetting the `b:livegrid` control.

#### Implementation

```
<div style="position:relative;width:500px; height:30px;">
  <b:livegrid-listener b:listento=".../~*[1]" style="padding:5px;
position:absolute;right:0px;" />
</div>
<b:livegrid b:datasource="datasource.php?myvalue" b:stylesheet="stylesheet.xsl"
b:totalattribute="total" b:loadatonce="24" />
```

## b:modal

Displays a modal dialog box on top of all other elements. The user cannot interact with the underlying document until the dialog is closed. You can open the modal dialog by triggering (using the `trigger` command) the `open` event, and close it by triggering the `close` event.

#### Implementation

In the following example, clicking the link opens a modal dialog, disabling interaction with

the rest of the interface until the modal is closed. Close the modal by clicking the link, or the close icon in the modal top-right hand corner.

```
<div>
  <p>
    <a b:action="trigger" b:event="open" b:target="id('modal')">What is The Godfather
about?</a>
  </p>
  <b:modal id="modal">
    <b:modalhead>The Godfather (1972)</b:modalhead>
    <b:modalbody>
      <p>The Godfather is a film adaptation of the novel of the same name (see The
Godfather novel) written by Mario Puzo, directed by Francis Ford Coppola and starring Marlon
Brando and Al Pacino.</p>
      <a b:action="trigger" b:event="close" b:target="id('modal')">Close the Modal</a>
    </b:modalbody>
  </b:modal>
</div>
```

## b:modalbody

Contains the body of a modal dialog.

---

**Attributes**

**b:innerstyle**

Use the standard CCS properties and values to style the content of a Back-base control, for example `b:innerstyle="background-color: black; color: white".`

---

**Parent Tags**

**b:modal**

---

**Child Tags**

You can define any BXML (and HTML) elements as children of this tag.

## b:modalhead

Contains the head of a modal dialog. (Allowed child tags are restricted by the size of the head element.)

---

**Parent Tags**

**b:modal**

---

**Child Tags**

Child tags are not allowed in this tag.

## b:navbody

The body element of a `b:navbox` in which you define the content that is displayed when the `b:navhead` is selected.

---

**Child Tags**

You can define any BXML (and HTML) elements as children of this tag.

---

**Parent Tags [Required]**

**b:navbox**

A control used for showing and hiding levels of sub-navigation and further detail. A `b:navbox` contains a single `b:navhead` and `b:navbody` element; clicking the head expands or collapses the body element in which the content is defined. The navboxes are block level, meaning that they are presented beneath one other and occupy the entire width of an area. Navboxes are designed for secondary navigational purposes; each navbox is a separate element that typically contains additional information loosely related

to the others.

#### Attributes

##### b:open

Set to `true` to open/expand the element by default.

#### Child Tags [Required]

##### b:navbody , b:navhead

## Implementation

The following is an example of a navbox:

```
<div style="background-color:#3399FF; width: 280px; padding: 10px; height: 300px">
  <b:navbox style="width: 250px;">
    <b:navhead>System Tasks</b:navhead>
    <b:navbody>
      <a>View System Tasks</a>
      <br />
      <a>Add or remove programs</a>
      <br />
      <a>Change a setting</a>
    </b:navbody>
  </b:navbox>
  <br />
  <b:navbox style="width: 250px; " b:open="true">
    <b:navhead>Other Places</b:navhead>
    <b:navbody>
      <a>My Network Places</a>
      <br />
      <a>My Documents</a>
      <br />
      <a>Details</a>
    </b:navbody>
  </b:navbox>
  <br />
  <b:navbox style="width: 250px;">
    <b:navhead>Details</b:navhead>
    <b:navbody>
      <a>My Computer</a>
    </b:navbody>
  </b:navbox>
</div>
```

## b:navhead

The head element of a `b:navbox`. Clicking the head displays the content defined in the `b:navbody`. (Allowed child tags are restricted by the size of the head element.)

#### Child Tags

Child tags are not allowed in this tag.

#### Parent Tags [Required]

##### b:navbox

## b:navpanel

A control that the user can expand or collapse by clicking the head and can be used for the main navigation of a web application. The navpanel assumes the full the height and width of its parent element. The `b:navpanelbody` contains the content. A `b:navpanel` typically has multiple `b:navpanelhead` and `b:navpanelbody` elements, only one of which can be selected at any time: selecting one automatically deselects the other.

#### Child Tags [Required]

##### b:navpanelbody , b:navpanelhead

## Implementation

```
<div style="height: 500px; width: 200px">
```

```

<b:navpanel b:singular="strict">
  <b:navpanelhead>The Seven Samurai (1954)</b:navpanelhead>
  <b:navpanelbody>
    <p>A movie by Akira Kurosawa starring Takashi Shimura and Toshiro Mifune.</p>
  </b:navpanelbody>
  <b:navpanelhead>2001: A Space Odyssey (1968)</b:navpanelhead>
  <b:navpanelbody>
    <p>An influential science fiction film directed by Stanley Kubrick.</p>
  </b:navpanelbody>
  <b:navpanelhead b:state="selected">Life Is Beautiful (1997)</b:navpanelhead>
  <b:navpanelbody>
    <p>A Italian language film which tells the story of an Italian Jew, Guido Orefice (played by Roberto Benigni), who lives in a romantic fairy tale, but must learn how to use that dreamy quality to survive a concentration camp with his young son Joshua (played by Giorgio Cantarini).</p>
  </b:navpanelbody>
</b:navpanel>
</div>

```

## b:navpanelbody

The body element of an item in a `b:navpanel` in which you define the content that is displayed when its `b:navpanelhead` is selected.

### Child Tags

You can define any BXML (and HTML) elements as children of this tag.

### Parent Tags [Required]

`b:navpanel`

## b:navpanelhead

The head element of an item in a `b:navpanel`. Clicking the head displays the content defined in the `b:navpanelbody`.

### Parent Tags [Required]

`b:navpanel`

## b:option

Defines an option in drop-down list. The `b:select` control contains child elements `b:option` tags that define the list of options.

### Attributes

`b:value`

A string containing a stated value.

### Parent Tags [Required]

`b:select`

## b:orgchart

Used for defining a hierarchical structure, for example to define an organization. The `b:orgchart` is the root of the hierarchy, and contains nested levels of `b:subordinate` elements. The tags have a `b:label` attribute.

### Attributes

`b:label`

Specify a String value that displays text in a set location for a control.

### Child Tags [Required]

`b:subordinate`

## Implementation

```

<b:orgchart b:label="Acme Organization">
  <b:subordinate b:label="Human Resources" />

```

```

<b:subordinate b:label="Development">
  <b:subordinate b:label="Research" />
  <b:subordinate b:label="Testing">
    <b:subordinate b:label="Mike">
      <b:subordinate b:label="Sarah"></b:subordinate>
    </b:subordinate>
  </b:subordinate>
</b:subordinate>
<b:subordinate b:label="Architecture" />
<b:subordinate b:label="Technical Support" />
<b:subordinate b:label="Technical Publications">
  <b:subordinate b:label="Technical Writing" />
  <b:subordinate b:label="Publishing" />
  <b:subordinate b:label="Training">
    <b:subordinate b:label="Instructor-Lead" />
    <b:subordinate b:label="Online" />
  </b:subordinate>
</b:subordinate>
</b:orgchart>

```

## b:page

A control used for controlling the paging mechanism, specifically the browsing functionality, of a data control (for example [b:datagrid](#) or [b:listgrid](#)).

Paging is essential for large data sets. It allows you the limit the number of records of the used data set retrieved and displayed at any time, and to scroll backwards and forwards through the data set. The [b:page](#) control defines the actual links for browsing through the data set; you need to define attributes on the data control for defining, for example, page size and number of retrieved records.

The [b:page](#) control triggers specific events that you can use to customize and extend the control:

- [select-page](#) event occurs when a [b:page-item](#) in a [b:page](#) control is clicked. It is used as part of implementing a paging mechanism.
- [draw-pagers](#) event instructs all pagers observing the control to be redrawn. It is triggered by the [select-page](#) event handler.
- [disable-pagers](#) event disables all pagers observing the control. It is triggered at the start of the [select-page](#) event handler.
- [enable-pagers](#) event enables all pagers observing the control. It is triggered at the end of the [select-page](#) event handler.

When a [select-page](#) event is triggered, the events are triggered in the order; [disable-pagers](#), [draw-pagers](#), [enable-pagers](#).

### Attributes

#### [Required] [b:observe](#)

Connect an element to an observed element. The value is an XPath expression pointing to the element to be observed. You can respond to events that are triggered on the observed element using the [observe-](#) prefix before the event name, for example [b:on="observe-command"](#).

#### Child Tags [Required]

[b:page-item](#) , [b:page-numbers](#)

## Implementation

```

<b:page b:observe="id('mydatagrid')">
  <b:page-item b:page-type="first">First</b:page-item>
  <b:page-item b:page-type="previous">Back</b:page-item>
  <b:page-numbers b:pagelinks="5" />
  <b:page-item b:page-type="next">Next</b:page-item>
  <b:page-item b:page-type="last">Last</b:page-item>
</b:page>

```

## b:page-item

Defines the links used to browse through the data set of a [b:page](#) control.

### Attributes

#### [Required] b:page-type

Define a position in the data set where the page item springs to.

Values:  
first  
last  
next  
previous

### Parent Tags [Required]

[b:page](#)

## b:page-numbers

Defines the number of pages that are displayed as clickable links. For example, if you want 5 pages to be available in a data set of 50 records, and the currently selected record is 25, the following links are displayed: **23 24 25 26 27**.

### Attributes

#### [Required] b:page-links

Define how many pages of the data set are displayed as clickable links.

### Parent Tags [Required]

[b:page](#)

## b:property

Defines a property inside the detailviewer control. The content can consist of any HTML markup.

### Attributes

#### [Required] b:label

Specify a String value that displays text in a set location for a control.

#### b:style

Use the standard CCS properties and values to style the inner elements of a Backbase control, for example `b:style="width: 100px"`. Note that the standard HTML `style` attribute sets the style for the outer part of the control.

### Child Tags

Child tags are not allowed in this tag.

### Parent Tags [Required]

[b:detailviewer](#)

## b:select

Provides a drop-down list. The [b:select](#) contains child elements [b:option](#) tags that define the list of options in the drop-down list. You can change the value, when the cursor is in the input field, by pressing the up and down arrow keys, and you can press the spacebar to cancel. The [b:select](#) tag contains child [b:option](#) tags that define an item in the list.

The [b:select](#) tag `b:name` attribute is the name of the input field that is sent on a form submit, and the `b:value` attribute sets the initial field value. The [b:option](#) tag `b:value` attribute is the value of a value-representation pair.

Due to a problem with Internet Explorer, you cannot use % to indicate width size (`b:width`) of a [b:select](#).

## Attributes

### [Required] b:name

Specify a logical unique name with which to identify an element.

### b:value

A string containing a stated value.

### b:text

Sets the text to display.

### b:width

Specify a percentage or pixel value specifying the width of an element.

## Child Tags [Required]

### b:option

## Implementation

```
<b:select b:name="county" b:value="Sweden" b:width="30%">
  <b:option b:value="1">England</b:option>
  <b:option b:value="2">France</b:option>
  <b:option b:value="3">Germany</b:option>
  <b:option b:value="4">South Africa</b:option>
  <b:option b:value="5">Italy</b:option>
  <b:option b:value="6">Poland</b:option>
  <b:option b:value="7">Sweden</b:option>
  <b:option b:value="8">Denmark</b:option>
  <b:option b:value="9">Netherlands</b:option>
  <b:option b:value="10">Belgium</b:option>
</b:select>
```

## b:separator

A divider for use in a statusbar.

## Parent Tags [Required]

### b:statusbar

## b:sidebar

Defines a tab in a b:sidebarbox.

## Attributes

### [Required] b:img

Specify the location of an image that is used on the tab of the sidebar. The image should contain text that is displayed vertically to identify the contents of the tab.

## Child Tags

You can define any BXML (and HTML) elements as children of this tag.

## Parent Tags [Required]

### b:sidebarbox

## b:sidebarbox

A container for a tabbed control that you can position vertically on the left-hand side or right-hand side of the screen. It is partly visible when not active, and fully visible when active. Each tab of the control is defined using b:sidebar elements.

## Attributes

### b:autoclose

Specify that the sidebarbox does not close when deselected. You can close the sidebarbox by clicking the currently open tab. The default is true.

### b:width

Specify a percentage or pixel value specifying the width of an element.

### b:height

Specify a percentage or pixel value for the height of an element.

**b:open**

Set to `true` to open/expand the element by default.

**b:top**

Specify a percentage or pixel value that sets the top position from the top edge of the containing element. Negative values are allowed.

**b:side**

Specify the whether the `b:sidebarbox` is positioned on the left or right-hand side of the screen. The default is `left`.

Values:

`left`

`right`

**Child Tags [Required]****b:sidebar****Implementation**

```
<b:sidebarbox b:width="400" b:height="250" b:top="100" b:auto-close="false" b:side="left">
  <b:sidebar b:img="/Backbase/controls/backbase/b-sidebarbox/menu.gif">Lorem Ipsum is
simply dummy text of the printing and typesetting industry....</b:sidebar>
  <b:sidebar b:img="/Backbase/controls/backbase/b-sidebarbox/tools.gif">It is a long
established fact that a reader will be distracted by the readable content of a page when
looking at its layout.</b:sidebar>
  <b:sidebar b:img="/Backbase/controls/backbase/b-sidebarbox/brushes.gif">Contrary to
popular belief, Lorem Ipsum is not simply random text.</b:sidebar>
</b:sidebarbox>
```

**b:spacer**

An invisible element that is used to separate elements. The spacer is an alternative implementation of spacer images found in most web sites. Spacers allow you to separate items to pixel precision.

**Attributes****b:width**

Specify a percentage or pixel value specifying the width of an element.

**b:height**

Specify a percentage or pixel value for the height of an element.

**Implementation**

```
<p>The following text has spacers inside.
  <br />
  Hello
  <b:spacer b:width="200px" b:height="150px" />
there!
  <b:spacer b:width="3em" />
How
  <b:spacer b:width="10px" />
are
  <b:spacer b:width="2px" />
you?
</p>
```

**b:spinner**

An element that displays up and down arrows in a text input type; you can go up and down through a range of numbers by pressing the arrows with the mouse or by using arrow keys. The current value of which is displayed in the element. The control triggers `loop-forward` and `loop-backwards` events when the user reaches the end, or beginning, of the possible values. When its value is changed, it also triggers the `change` event. If the user types a value in the field, the values are validated as follows:

- If the value is lower than the `b:start`, the value of the `b:start` is used
- If the value is greater than the `b:end`, the value of `b:end` is used

- If the value is not a number, the value reverts to the last known valid value

---

**Attributes****b:step**

Specify the step size which a value is incremented or decremented from a lower bound to an upper bound, and vice versa. The default is 1.

**b:start**

Set the minimum lower bounding value. The default is 1.

**b:end**

Sets the maximum upper bounding value. The default is 100.

**b:loop**

Set to `false` to prevent the control from looping when it reaches the end of its upper or lower bound. The default is `true`.

**b:value**

A string containing a stated value.

**b:style**

Use the standard CCS properties and values to style the inner elements of a Backbase control, for example `b:style="width: 100px"`. Note that the standard HTML `style` attribute sets the style for the outer part of the control.

---

**Implementation**

The following is an example of a spinner control:

```
<div>Spinner control:
  <b:spinner b:start="-5" b:value="4" b:step="1" b:end="5" b:style="width: 60px" />
</div>
```

---



---

**b:statusbar**

An information bar usually presented at the bottom of a page. It can be divided into several separately accessible sections.

---

**Child Tags****b:separator**

---

**Implementation**

The following is an example of a statusbar:

```
<b:statusbar style="position: absolute; bottom:0; left:0; width:100%;">Online!
  <b:separator />
Offline!
</b:statusbar>
```

---



---

**b:subordinate**

Used for defining a hierarchical structure, for example to define an organization. The `b:orgchart` is the root of the hierarchy, and contains nested levels of `b:subordinate` elements. The tags have a `b:label` attribute.

---

**Attributes****b:label**

Specify a String value that displays text in a set location for a control.

---

**Child Tags****b:subordinate**

---

**Parent Tags [Required]****b:orgchart**

---

**b:tab**

Contains the contents of a tab in a `b:tabbox`.

---

**Attributes**

**[Required] b:label**

Specify a String value that displays text in a set location for a control.

**b:url**

Specify the path of the file to load, relative to the startup file of your Backbase application.

**b:dirty**

Specify `b:dirty="true"` to indicate the content requires reloading the next time the element is opened or selected. After reload, it is automatically set back to `b:dirty="false"`.

**Child Tags**

You can define any BXML (and HTML) elements as children of this tag.

**Parent Tags [Required]****b:tabbox****b:tabbox**

Provides a tabbed interface, each tab being defined within a `b:tab` tag. The `b:tabbox` control supports lazy loading: when you specify the tab contents using a `b:url`, the contents are only loaded at the moment a user clicks the tab for the first time. When a `b:url` is specified, the `populate` event - that is triggered when a tab is selected - is used to populate the tab with data.

**Child Tags [Required]****b:tab****Implementation**

```
<b:tabbox style="height: 100%">
  <b:tab b:label="2001" b:url="data/2001_short.xml" />
  <b:tab b:label="Seven Samurai" b:url="data/seven_short.xml" />
  <b:tab b:label="Life is Beautiful" b:state="selected" b:url="data/bella_short.xml" />
  <b:tab b:label="The Godfather" b:url="data/godfather_short.xml" />
</b:tabbox>
```

**b:taskbar**

A taskbar is a windowmanager that is displayed as a tabbed control and is used to manage the windows that are opened and closed within a `b:windowarea` container element. The windowmanager automatically receives event calls from window controls to register themselves with the windowmanager. The `b:startbutton` element is used to define the first tab in the taskbar.

**Attributes****b:windowarea**

Specify the `b:windowarea` in which the windows, when opened, are registered in the taskbar.

**Implementation**

The following example shows skeleton code for defining a `b:taskbar`. For more information, see `b:windowarea`.

```
<b:panelset b:rows="* 28px">
  <b:panel>
    <b:windowarea id="windowarea" style="height:100%;width:100%;border:0;">
      <!--add your windows here; when opened, they are registered in the taskbar-->
  </b:windowarea>
  </b:panel>
  <b:panel>
    <b:taskbar id="taskbar" b:windowarea="id('windowarea')">
      <b:startbutton />
    </b:taskbar>
  </b:panel>
</b:panelset>
```

```
</b:panel>
</b:panelset>
```

## b:tilelist

Tiles its contents, defined within the `b:tilelist-body` element, into columns or rows. You can specify the `b:url` attribute to load the contents from an external file, or specify the contents by specifying an inline `b:tilelist-body` element. The tilelist tiles the child elements of the `b:tilelist-body` element.

The control also has properties for defining a paging mechanism, used in combination with the `b:page` control. You need to implement the paging on the server-side.

When you use the `b:tilelist`, ensure that ancestor elements do not have the `style="display:none"` property set. (The control can experience problems when being drawn if this property is set.)

### Attributes

#### `b:orientation`

Specify the orientation of elements displayed in a tile formation. The default is `cols`.

Values:  
`cols`  
`rows`

#### `b:url`

Specify the path of the file to load, relative to the startup file of your Backbase application.

#### `b:rowsmargin`

Specify the spacing between rows using using `%`, `px`, or `em`.

#### `b:colsmargin`

Specify the spacing between columns using `%`, `px`, or `em`.

#### `b:page-size`

Define the number of records displayed in every page.

#### `b:page-number`

Define the initial page that is displayed when the page is constructed. When you navigate through the pages, the value is dynamically updated to indicate which page is currently displayed.

#### `b:page-cache`

Define the number of pages retained in memory (0 or 1 are identical).

#### `b:pages-in-total`

Read-only attribute that is calculated at runtime, if the control is connected to a paged data set, by dividing the `b:items-in-total` attribute by the `b:page-size` attribute.

#### `b:redraw`

Redraws the tilelist when the window is resized. The default value is `true`.

### Child Tags

[Required] `b:tilelist-body`

## Implementation

In the following example, the tilelist tiles the children of the `b:tilelist-body` element that is loaded using the `b:url` attributes.

```
<style type="text/css">      .mytile {      position: absolute;      width: 80px;
height: 45px;      margin: 0 10px 10px 0;      padding: 5px;      border: 1px solid #345;
} </style>
<div style="height: 50px; width: 200px">
  <b:page b:observe="id('contacts-tilelist')">
    <b:page-item b:page-type="first">First</b:page-item>
    <b:page-item b:page-type="previous">Back</b:page-item>
    <b:page-numbers b:page-links="3" />

```

```

        <b:page-item b:page-type="next">Next</b:page-item>
        <b:page-item b:page-type="last">Last</b:page-item>
    </b:page>
</div>
<b:tilelist style="height:200px; width: 500px" id="contacts-tilelist" b:orientation="rows"
b:page-size="30" b:page-number="1" b:page-cache="3" b:redraw="true"
b:url="tilelist.xml"></b:tilelist>
```

Where the contents of the `b:url="tilelist.xml"` file is as follows (note that loaded files must include namespace declarations for its content):

```

<?xml version="1.0"?>
<b:tilelist-body xmlns="http://www.w3.org/1999/xhtml" xmlns:b="http://www.backbase.com/b"
xmlns:s="http://www.backbase.com/s" b:items-in-total="100" b:dirty="true">
<div class="mytile">content 0</div>
<div class="mytile">content 1</div>
<div class="mytile">content 2</div>
<div class="mytile">content 3</div>
<div class="mytile">content 4</div>
<div class="mytile">content 5</div>
<div class="mytile">content 6</div>
<div class="mytile">content 7</div>
<div class="mytile">content 8</div>
<div class="mytile">content 9</div>
<div class="mytile">content 10</div>
<div class="mytile">content 11</div>
<div class="mytile">content 12</div>
<div class="mytile">content 13</div>
<div class="mytile">content 14</div>
<div class="mytile">content 15</div>
<div class="mytile">content 16</div>
<div class="mytile">content 17</div>
<div class="mytile">content 18</div>
<div class="mytile">content 19</div>
<div class="mytile">content 20</div>
<div class="mytile">content 21</div>
<div class="mytile">content 22</div>
<div class="mytile">content 23</div>
<div class="mytile">content 24</div>
<div class="mytile">content 25</div>
<div class="mytile">content 26</div>
<div class="mytile">content 27</div>
<div class="mytile">content 28</div>
<div class="mytile">content 29</div>
</b:tilelist-body>
```

## b:tilelist-body

Defines the content of a `b:tilelist` control. The tilelist tiles the child elements of the `b:tilelist-body` element. The children must have `position:absolute`.

### Attributes

#### `b:items-in-total`

The total number of records returned in the paged data set. The value is used to calculate the total number of pages, and should be placed on the `b:tilelist-body` every time the server serves it so that the tilelist knows the current size of the data set.

#### `b:dirty`

Specify `b:dirty="true"` to indicate the content requires reloading the next time the element is opened or selected. After reload, it is automatically set back to `b:dirty="false"`.

### Parent Tags

[Required] `b:tilelist`

## b:toolbar

Presents a navigation bar. The elements that are placed in the toolbar are only limited in

height. The toolbar occupies the full width of an area.

#### Child Tags

[Required] **b:toolbaritem**

## b:toolbaritem

Presents an item in a toolbar. These can be any kind of element as long as the height of this bar is taken into account. Usually the child elements will either be images or form elements.

#### Child Tags

You can define any BXML (and HTML) elements as children of this tag.

#### Parent Tags [Required]

[Required] **b:toolbar**

## b:tree

Presents a tree structure defined using nested **b:tree** elements. By default, a **b:tree** node that does not contain child nodes is a leaf node and is rendered using a different image. You can force a node to be a folder or leaf by using the **b:leaf** or **b:folder** attributes.

#### Attributes

##### **b:label**

Specify a String value that displays text in a set location for a control.

##### **b:dirty**

Specify `b:dirty="true"` to indicate the content requires reloading the next time the element is opened or selected. After reload, it is automatically set back to `b:dirty="false"`.

##### **b:url**

Specify the path of the file to load, relative to the startup file of your Backbase application.

##### **b:lazy-loading**

Set to `true` to add support for lazy loading. When a user clicks a node for the first time, the data for the node is loaded. Lazy loading means that data is only loaded when it is needed. You can force a tree node to reload by defining a `refresh` event handler for the node, for example `b:action="trigger" b:event="refresh" b:target=""` (if no target is specified, the default is the context node).

You can only use the attribute on the root tag of the tree structure.

##### **b:multiroot**

When set to `true`, the root element is used as a container only. All direct children are visible, making it look like the tree has more than one root element.

You can only set the attribute on the root tag of the tree structure.

##### **b:image-when-open**

Specify a url to an image that is used when this node is opened. By default, the node uses a standard image defined for the control.

##### **b:image-when-closed**

Specify a url to the image that is used when this node is closed. By default, the node uses the standard image defined for the control.

##### **b:leaf**

Set to `true` to display the node as a leaf node.

##### **b:folder**

Set to `true` to display the node as a folder node.

#### Child Tags

**b:tree**

## b:window

Creates a window that you can move, similar to windows in desktop applications. The child tags **b:windowhead** and **b:windowbody** define the head and body of the window. You can close an active window with the keys Shift+F4.

If the window is maximized, its width and height is the same as the first parent element with a CSS position style of 'absolute' or 'relative'.

### Attributes

#### b:open

Set to `true` to open/expand the element by default.

#### b:maximized

Specify whether the window is maximized when opened. The default is `false`.

#### b:minimized

Specify whether the window is minimized when opened. The default is `false`.

#### b:windowbuttons

Specify the buttons that appear in the **b:windowhead**. You can use several values by comma or space separating the values.

Values:

close  
maximize  
minimize  
none

### Child Tags

#### b:windowbody , b:windowhead

## Implementation

In the following example, click the links to open the windows.

```
<div style="position: relative; height: 100%; overflow:visible; border : 1px solid red;">
    <a b:action="trigger" b:event="open" b:target="id('win3')">Open window 1</a>
    |
    <a b:action="trigger" b:event="open" b:target="id('win4')">Open window 2</a>
    <b:window id="win3" b:open="true" style="top:100px; left:20px;">
        <b:windowhead>Life Is Beautiful (La Vita è Bella)</b:windowhead>
        <b:windowbody>A 1997 Italian language film which tells the story of an Italian
Jew...</b:windowbody>
    </b:window>
    <b:window id="win4" b:open="false" style="top:150px; left:100px;">
        <b:windowhead>2001: A Space Odyssey</b:windowhead>
        <b:windowbody>An influential science fiction film directed by Stanley Kubrick,
notable for combining episodes contrasting high levels of scientific and technical realism
with transcendental mysticism.</b:windowbody>
    </b:window>
</div>
```

## b:windowarea

Serves as a container for **b:window** elements. It is used in combination a **b:taskbar**, which acts as a windowmanager for windows defined within the windowarea.

### Child Tags

#### b:window

## Implementation

```
<s:execute>
    <s:task b:action="show" />
</s:execute>
<b:panelset b:rows="* 28px">
    <b:panel>
        <b:windowarea id="windowarea" style="height:100%;width:100%;border:0;">
            <a b:action="trigger" b:event="open" b:target="id('window1')"
style="position:absolute;left:65px;top:50px;">Weather</a>
            <a b:action="trigger" b:event="open" b:target="id('window2')"
style="position:absolute;left:65px;top:88px;">Sport</a>
```

```

<b:window id="window1" style="left:10px;top:10px;width:370px;height:290px;">
    <b:windowhead>Weather</b:windowhead>
    <b:windowbody b:innerstyle="padding:0;overflow:hidden;">
        <iframe
            style="border-style:none;margin:0;padding:0;width:100%;height:100%"
            src="http://weather.yahoo.com/" />
    </b:windowbody>
</b:window>
<b:window b:resize="" id="window2"
    style="left:50px;top:50px;width:540px;height:580px;">
    <b:windowhead>BBC</b:windowhead>
    <b:windowbody b:innerstyle="padding:0;overflow:hidden;">
        <iframe
            style="border-style:none;margin:0;padding:0;width:100%;height:100%"
            src="http://www.bbc.co.uk/sport" />
    </b:windowbody>
</b:window>
</b:windowarea>
</b:panel>
<b:panel>
    <b:taskbar id="taskbar" b:windowarea="id('windowarea')">
        <b:startbutton />
    </b:taskbar>
</b:panel>
</b:panelset>

```

## b:windowbody

Defines the body of a window tag.

---

### Attributes

#### [b:innerstyle](#)

Use the standard CCS properties and values to style the content of a Backbase control, for example `b:innerstyle="background-color: black; color: white"`.

---

### Child Tags

You can define any BXML (and HTML) elements as children of this tag.

---

### Parent Tags [Required]

#### [b:window](#)

## b:windowhead

Defines the heading of a window control.

---

### Attributes

#### [b:icon](#)

Specify the path to an image file that contains an icon that is displayed in the top left-hand corner.

---

### Child Tags

Child tags are not allowed in this tag.

---

### Parent Tags [Required]

#### [b:window](#)

## Generic Attributes

### b:action

Specify the command to execute when the element's command event fires, for example when it is clicked. Each command has specific attributes. See the command section for an overview of all the commands.

#### Implementation

In the following example, clicking the `a` element results in an alert message displaying the text "Hi!".

```
<a b:action="alert" b:value="Hi!">Click here to show the alert.</a>
```

For more information on Events and Commands, see the [Manual.pdf](#).

### b:behavior

Attaches a *behavior*, defined elsewhere within `s:behavior` tags, to the element. The value is the same as the `b:name` value of the behavior.

#### Implementation

In the following example, the `a` (link) element uses the behavior called `mybehavior`. Clicking the link results in a popup alert message displaying the text "Hi!".

```
<a b:behavior="mybehavior">Click Me!</a>
<s:behavior b:name="mybehavior">
  <s:event b:on="command">
    <s:task b:action="alert" b:value="Hi!" />
  </s:event>
</s:behavior>
```

### b:columnmode

When you resize a table, by default the table grows larger and smaller as you resize the columns. When you specify `b:columnmode="fixed"`, the table is fixed and when you resize a column only the column next to it grows smaller or larger.

Values:  
fixed

### b:cursor

Control the type of cursor that should be used for a specific section of the site. Typically, if you use a `div` tag with a `b:action` attribute, you also want to assign the cursor pointer to inform the user that the element is clickable.

You can, for example, set a default cursor for the entire web application on the first content node in your web application. This suppresses the browser's default behavior to change cursors when hovering over normal text, therefore resulting in a more consistent application.

The cursor appearance can vary from browser to browser.

Values:  
crosshair  
default  
e-resize  
help  
move  
ne-resize  
n-resize  
nw-resize

```
pointer
text
wait
```

## Implementation

In the following example, when the mouse moves over the HTML element, the normal mouse pointer changes to a hand-type mouse pointer. Clicking the `p` element results in a popup alert message displaying the text "test!".

```
<span b:action="alert" b:value="test!" b:cursor="pointer">Hover to show 'you can click here'
pointer cursor.</span>
```

## b:disabled

Disable an element. No user input events can be triggered on an element that is disabled, only system events. You can enable a disabled element using the `set` or `enable` commands, or by using the `s:setatt` tag.

Values: true | false

## b:drag

Makes the element draggable. The value of the attribute is a logical string identifier that defines the *drag class*. The element can then be dropped into another element which has a `dragreceive` attribute, the value of which matches the drag class name of the draggable element.

When an element is dragged, the drag events `drag-enter`, `drag-leave`, `drag-drop`, `drag-receive`, and `drag-start` are triggered.

## Implementation

The following example illustrates a basic drag-and-drop implementation: elements extended with the `b:drag` attribute can be dragged (and dropped) into a receiving container (in this case, a `div` element extended with the `b:dragreceive` attribute).

Only elements extended with the `b:drag` attribute can be the gripping point for the drag and drop operation.

```
<p>Drag the elements from one container to the other...</p>
<div>Container One</div>
<div b:dragreceive="dndGroup" style="width:100px;height:50px;border:1px solid #B3B2B2;">
    <div b:drag="dndGroup" style="width:100%">
        <div style="background-color:#808080; text-align:center; color:white; margin:1px;">
            Item</div>
    </div>
    <div b:drag="dndGroup" style="width:100%">
        <div style="background-color:#808080; text-align:center; color:white; margin:1px;">
            Item</div>
    </div>
    <div b:drag="dndGroup" style="width:100%">
        <div style="background-color:#808080; text-align:center; color:white; margin:1px;">
            Item</div>
    </div>
</div>
<div>Container Two</div>
<div b:dragreceive="dndGroup" style="width:100px; height:50px; border:1px solid #B3B2B2;"></div>
```

By default, selecting an image behaves in the same way as selecting normal text; the generic windows clipboard is used, so when start dragging you can drag-and-drop the image outside the browser application. You can prevent this behavior by adding the attribute `b:textselect="false"` on the draggable element containing the image: For example:

```
<div b:textselect="false" b:drag="a" b:dragmode="real" id="1">
  
</div>
<div b:dragreceive="a" style="border: red solid 1px; position: absolute; top: 10px; left: 100px; width:100px; height:100px">Drop here</div>
```

It is recommended to use `b:textselect="false"` when using `b:dragmode="symbol"` because Mozilla has a known problem with firing mouse events when ancestor elements have `style` property `overflow` that is set to `auto`, `hidden`, or `scroll`.

## b:dragconstraint

Set an element as the constraining element for dragging. The dragged element cannot be dragged outside of the constraining element. If the dragged element is not inside the constraining element at the start of the drag, the dragged element snaps to the closest position inside the constraining element.

Values: The attribute accepts an XPath statement.

## b:dragconstrainttype

Specify the constraint type for dragging. When applying a `b:dragconstraint`, the default behavior is that you cannot drag the border of a dragged element outside the constraining element. You can also make the mouse cursor the constraining type instead of the border. You can then drag the element outside of the constraining element as long as the mouse cursor is still inside the constraining element.

Values:

`border`  
`mouse`

## b:draggroup

Group multiple elements in one drag. Place this attribute on the element you want to drag as a whole and place the `drag` attribute on one or more of its children with a matching class as value.

### Implementation

The following example illustrates a basic drag and drop implementation: an element extended with the `b:draggroup` attribute defines a node-set to be dragged (and dropped) into a receiving container (in this case, `div` elements extended with the `b:dragreceive` attribute).

Note that only the two `div` elements extended with the `b:drag` attribute can be the gripping point for the drag and drop operation.

Drag the elements from one container to the other.

The light grey element can not instantiate a drag but will be taken along with the rest of the group.

```
<div>Container One</div>
<div b:dragreceive="dndGroup" style="width:100px;height:50px;border:1px solid #B3B2B2;">
  <div b:draggroup="dndGroup">
    <div b:drag="dndGroup" style="width:100%">
      <div style="background-color:#808080; text-align: center; color:white; margin:1px;"> Item</div>
    </div>
    <div b:drag="dndGroup" style="width:100%">
      <div style="background-color:#808080; text-align:center; color:white; margin:1px;"> Item</div>
```

```

        </div>
        <div style="width:100%">
            <div style="background-color:#C0C0C0; text-align:center; color:white;
margin:1px;"> Item</div>
        </div>
    </div>
<div>Container Two</div>
<div b:dragreceive="dndGroup" style="width:100px; height:50px; border:1px solid
#B3B2B2;"></div>

```

## b:dragmode

Specify the appearance of the element while it is being dragged. If you are using a **draggroup**, place the **b:dragmode** on the element with the **b:draggroup** attribute, otherwise specify the attribute on the element that has the **b:drag** attribute defined.

- **outline** - (the default) the dragged item is an outline copy of the source element, the source element is unchanged
- **outline-hidden** - the same as **outline**, but the source element is **visibility:hidden**
- **outline-none** - the same as **outline**, but the source element is **display:none**
- **real** - the dragged element has the appearance of the source element
- **symbol** - allows you to define the dragmode using the **bpc\_dragSymbol** variable (see also the **drag-enter** or **drag-leave** events)

When using **b:dragmode="real"**, set **b:overflowfix="true"** on the **b:draggroup** element to prevent Internet Explorer crashing.

Values:

```

outline
outline-hidden
outline-none
real
symbol

```

## b:dragreceive

Specify that an element is a receiving container for drag-and-drop operations and defines the drag groups it accepts. It may contain a space separated set of drag group names values.

### Implementation

The following example illustrates how to restrict an element's drag and drop operation by using different **b:drag** attribute values and matching **b:dragreceive** attribute values.

The **div** extended with **b:drag="dndGroupX"** can only be dragged to the receiving container that is extended with **b:dragreceive="dndGroupX"** and then moved back to the original **div** parent element. Just like the Y element can only be dragged to the Y container and the Z element can only be dragged to the Z container.

```

<p>Drag the elements to their corresponding container.</p>
<div>X, Y and Z Container</div>
<div b:dragreceive="dndGroupX dndGroupY dndGroupZ" style="width:100px;height:50px;border:1px
solid #B3B2B2;">
    <div b:drag="dndGroupX" style="width:100%">
        <div style="background-color:#808080; text-align:center; color:white; margin:1px;">
            X</div>
    </div>
    <div b:drag="dndGroupY" style="width:100%">
        <div style="background-color:#808080; text-align:center; color:white; margin:1px;">
            Y</div>
    </div>

```

```

<div b:drag="dndGroupZ" style="width:100%">
    <div style="background-color:#808080; text-align:center; color:white; margin:1px;">
        Z</div>
    </div>
</div>
<div>X-Only Container</div>
<div b:dragreceive="dndGroupX" style="width:100px; height:20px; border:1px solid #B3B2B2;"></div>
<div>Y-Only Container</div>
<div b:dragreceive="dndGroupY" style="width:100px; height:20px; border:1px solid #B3B2B2;"></div>
<div>Z-Only Container</div>
<div b:dragreceive="dndGroupZ" style="width:100px; height:20px; border:1px solid #B3B2B2;"></div>

```

## b:eventblock

Disable all user input events on an element and its children. By default, the attribute is set to `false`. You can use the `eventblock` attribute to create modal dialogs where only the modal dialog responds to user input. If a user clicks on the area in which events are blocked, a `blocked-mousedown` event occurs instead of a `mousedown` event, allowing you to handle special cases. For more information, see [blocked-mousedown](#).

The `b:eventblock` does not work in IE when the events `dblclick` or `select` are triggered, and in Mozilla on radio button or select box elements.

Values: true | false

## b:focusgroup

Define a container for focusitem elements. When using keys to navigate through focus elements, you can only navigate through focusitems within a focusgroup. A focusgroup element can also be a focusitem, and can contain nested focusgroup elements.

You can cycle through focusgroup elements using the Tab and Shift+Tab keys. You can change these default keys using the `s:keys` tag.

### Implementation

The following example demonstrates visually the focus model. The focus system is hierarchical. Navigate through the groups and items and watch how the color indicates all focus containers upwards in the hierarchy. When the page is constructed, the color **Purple** is applied as the background. When events are triggered, the elements change color: **Green** indicates activated elements, **Red** indicates inactive elements, and **White** indicates the item that has focus (and is also active):

```

<s:behavior b:name="focusobject">
    <s:event b:on="construct">
        <s:setstyle b:padding="10px" b:background-color="#0099FF" />
    </s:event>
    <s:event b:on="active">
        <s:setstyle b:background-color="#00D856" />
    </s:event>
    <s:event b:on="inactive">
        <s:setstyle b:background-color="#D85500" />
    </s:event>
    <s:event b:on="focus">
        <s:setstyle b:background-color="white" />
    </s:event>
</s:behavior>
<div b:behavior="focusobject" b:focusroot="true">
    <p>This code demonstrates the focus model; It contains two focusgroups, 1 and 2.
    Focusgroup 1 contains focusitems 1.1 and 1.2. Focusgroup 2 contains focusitems 2.1, 2.2, 2.3
    and 2.4. You can set focus with the
    <b>Keyboard</b>

```

```

or the
    <b>mouse</b>
.

    </p>
    <p>The tab and arrow keys are used for keyboard navigation:</p>
    <ul>
        <li>Use the
            <b>Tab</b>
        , or
            <b>Tab+Shift</b>
        , key to navigate between groups (the focusgroup remembers the focusitem that last had focus
        within a group.)
            </li>
            <li>Use the up and down
                <b>arrow keys</b>
        change focus between items within a group.
            </li>
        </ul>
        <div b:behavior="focusobject" b:focusgroup="true">Focusgroup 1
            <div b:behavior="focusobject" b:focusgroup="false" b:focusitem="true">Focusitem
        1.1</div>
            <div b:behavior="focusobject" b:focusgroup="false" b:focusitem="true">Focusitem
        1.2</div>
            </div>
            <div b:behavior="focusobject" b:focusgroup="true">Focusgroup 2
                <div b:behavior="focusobject" b:focusgroup="false" b:focusitem="true">Focusitem
        2.1</div>
                <div b:behavior="focusobject" b:focusgroup="false" b:focusitem="true">Focusitem
        2.2</div>
                <div b:behavior="focusobject" b:focusgroup="false" b:focusitem="true">Focusitem
        2.3</div>
                <div b:behavior="focusobject" b:focusgroup="false" b:focusitem="true">Focusitem
        2.4</div>
            </div>
        </div>
    </div>

```

Values: true | false

## b:focusindicator

Visualize the focusitem element by creating a dotted line around the element.

Values: true | false

## b:focusitem

Set the value to `true` to enable the element to get focus. HTML form elements are focusable by default.

You can cycle through focusitem elements using the arrow keys right and down, and left and up. You can change these default keys using the `s:keys` tag.

Values: true | false

## b:focusroot

Define the top most element within a focus hierarchy. When moving up and down in the focus hierarchy, you cannot go above the focusroot. You can have multiple focus roots inside your application.

You can cycle through focusroot elements using the Ctrl+Down and Ctrl+Right keys. You can change these default keys using the `s:keys` tag.

Values: true | false

## b:followstate

Link the state of one element to another by specifying an XPath expression that targets the element whose state you want to follow. At any time, an element in a Backbase ap-

plication can be in a state **selected**, or **deselected**. When the followed element is selected, the element whose **b:followstate** attribute targets the selected element also becomes selected, allowing you to synchronize these elements. Note that the **b:followstate** attribute is a more specific implementation of the same mechanism used in **b:observe**.

## Implementation

In the following example, the **a** and the **b:navbox** are interconnected. When you click on the **a**, which has a **select-deselect** toggle command on itself, the **b:followstate** attribute makes the **b:navbox** follow the **a**'s state, causing it to open and close.

```
<div>
    <a id="openclose" b:cursor="pointer">Open/Close All Boxes</a>
    <s:event b:on="click">
        <s:task b:action="select-deselect" b:target="id('openclose')" />
    </s:event>
</div>
<br />
<div>
    <b:navbox b:followstate="id('openclose')" style="width: 250px;">
        <b:navhead>The Seven Samurai</b:navhead>
        <b:navbody>
            <p>A movie by Akira Kurosawa starring Takashi Shimura and Toshiro Mifune. The film takes place in war-ridden 16th century Japan.</p>
        </b:navbody>
    </b:navbox>
</div>
<br />
<div>
    <b:navbox b:followstate="id('openclose')" style="width: 250px;">
        <b:navhead>2001: A Space Odyssey</b:navhead>
        <b:navbody>
            <p>An influential science fiction film directed by Stanley Kubrick.</p>
            <a href="http://kubrickfilms.warnerbros.com/">More Info</a>
        </b:navbody>
    </b:navbox>
</div>
<br />
<div>
    <b:navbox b:followstate="id('openclose')" style="width: 250px;">
        <b:navhead>Life Is Beautiful</b:navhead>
        <b:navbody>
            <p>A Italian language film which tells the story of an Italian Jew who uses his fantastical imagination to survive a concentration camp with his young son Joshua.</p>
        </b:navbody>
    </b:navbox>
</div>
```

Values: The attribute accepts an XPath statement.

## **b:lineconstraint**

When you specify **b:resizemode="line"**, you can also specify an element, using an XPath expression, from which the line must get its height or width from (depending on whether the resizing is vertical or horizontal).

Values:  
XPath

## **b:maxheight**

Specify the maximum height of an element in pixels, for example **400px**. You can use the attribute in combination with, for example, the **b:resize** attribute to limit resizing.

## **b:maxwidth**

Specify the maximum width of the element in pixels, for example: 400px. You can use the attribute in combination with, for example, the `b:resize` attribute to limit resizing.

## b:minheight

Specify the minimum height of an element in pixels, for example 300px. You can use the attribute in combination with, for example, the `b:resize` attribute to limit resizing.

## b:minwidth

Specify the minimum width of an element in pixels, for example 200px. You can use the attribute in combination with, for example, the `b:resize` attribute to limit resizing.

## b:observe

Connect an element to an observed element. The value is an XPath expression pointing to the element to be observed. You can respond to events that are triggered on the observed element using the `observe-` prefix before the event name, for example `b:on="observe-command"`.

### Implementation

This example illustrates the use of this attribute to synchronize events and commands: when you click on the first `div` element, the `div` extended with the `b:observe` attribute will receive the `command` event (`show-hide`) and execute the same tasks as well.

```
<p id="observed" b:cursor="pointer">
    <s:event b:on="command">
        <s:task b:action="show-hide" b:target="*[2]" />
    </s:event>
    <div>Click here to show and hide the
        <em>observed</em>
    element.
    </div>
    <div style="font-weight: bold; color: #04435E;">Observed element...</div>
</p>
<p b:observe="id('observed')" b:cursor="pointer">
    <s:event b:on="observe-command">
        <s:task b:action="show-hide" b:target="*[2]" />
    </s:event>
    <div>
        <em>Observing</em>
    element (will mimic observed element):
    </div>
    <div style="font-weight: bold; color: #CE0000;">...Observing Element</div>
</p>
```

Values: The attribute accepts an XPath statement.

## b:resize

Make the element resizable, enabling the user to make an element larger or smaller. You can resize a resizable element by selecting its edge with the cursor and, holding the mouse button down, drag the edge in the desired direction. You can add a `b:resize` attribute to virtually any XHTML tag or b tag. You must specify a `position: absolute` or `position: relative` for the resizable element.

Note that in some situations, resizing can be used incorrectly where resizing is not strictly possible. For example, trying to resize the following elements results in strange behavior: non-block level elements, non- relative or absolute positioned elements, and resize north of a non- absolute element.

**Values:**

all  
e  
horizontal  
n  
ne  
nw  
s  
se  
sw  
vertical  
w

## Implementation

In the following example, the window is resizable in all directions. A **b:minheight** and **b:minwidth** is also specified to limit the resizing (**b:minheight** and **b:minwidth**):

```
<div style="position: relative; height: 100%; overflow:visible; border : 1px solid red;">
    <a b:action="trigger" b:event="open" b:target="id('win3')">Open window 1</a>
    <b:window id="win3" b:open="true" style="position: absolute; top:25px; left:30px; "
    b:resize="all" b:minheight="100px" b:minwidth="100px">
        <s:event b:on="construct"></s:event>
        <b:windowhead>Life Is Beautiful (La Vita è Bella)</b:windowhead>
        <b:windowbody>
            <s:variable b:name="myTag" b:scope="tag" />
            <p>A 1997 Italian language film which tells the story of an Italian Jew...</p>
        </b:windowbody>
    </b:window>
</div>
```

## **b:resizeconstraint**

Prevent an element from being resized beyond the size of a specific element, for example its parent element. The value is an XPath statement indicating its containing element.

**Values:**

XPath

## **b:resizemode**

Define the appearance of the element that is being resized (has the **b:resize** attribute set):

- **line** draws a line on the side of the element you are resizing. Line resizing is only possible in the directions when resize is: N, E, S, or W, and tables can only be resized E, N, or S. It is the default mode for resizing tables and panelsets.  
You can use the XPath variable `$bpc_resizeLine` to indicate the styling that is applied when line-resizing. By default, it is set to `border:4px solid #AAA;`. See also: **b:lineconstraint**
- **outline** draws an outline of the element you are resizing (the same as when dragging an element). It is the default mode for all elements except tables and panelsets.
- **real** - redraws the element as it is being resized (can be very slow, depending on the element you are trying to resize).

**Values:**

line  
outline  
real

## **b:singular**

Impose the restriction on an element's child tag's that only one child can have, at any one time, the state **selected**. The first child is automatically selected if none of the chil-

dren has been explicitly assigned the selected state (`b:state="selected"`). By default, the value is set to `none` (no restriction).

#### Values:

`none`  
`strict`

## b:state

Set the default state of an element. An element in a Backbase application can be in the state **selected**, or in the state **deselected**. Most BXML tags show a visible reaction according to their state. For example, a `b:modal` window becomes active and visible when it is in the selected state.

#### Values:

`deselected`  
`selected`

### Implementation

The following example shows how to set a default state for a deck child element. Within a deck, the first child node be selected by default. You can override the default behavior by specifying the `selected` state on a different child node.

```
<b:deck class="simpleDeck">
  <div>1 This is a deck...</div>
  <div b:state="selected">2...a control to create...</div>
  <div>3...multi-paged documents.</div>
</b:deck>
<a b:action="previous" b:target="..></b:deck">&lt; Previous</a>
<a b:action="next" b:target="..></b:deck">Next &gt;</a>
```

## b:test

Specify a condition, expressed in XPath, that returns either true or false. For example, `b:test="@border = '0'"` checks whether the border attribute of the context node is set to 0; if it isn't, the code is not executed.

Values: The attribute accepts an XPath statement.

## b:textselect

Enables or disables text selection for all the children of an element (useful for tab headings, menus, buttons and trees).

Note that you cannot overrule `b:textselect` at a deeper level.

Values: true | false

## b:tooltiptext

Specify a value that is shown as a tooltip when the user hovers the mouse over the element.

## b:zsort

Move the element and place after its last sibling. When the element overlaps with its siblings, the element is rendered on top of the rest.

#### Values:

`false`  
`true`

## BXML Commands

### addclass

Adds the style class, specified using the `b:value` attribute, to the style classes used by the targeted element. The standard CSS rules apply concerning the priorities for which class overrules another, and do not depend on the order in which they occur on the element.

You can use this command on elements in the *BXML Space* and *HTML Space*.

#### Attributes

**[Required] b:value**

A string containing a stated value.

**b:target**

Specify an XPath expression pointing to the target element. The default value is `.` (self).

### Implementation

```
<style type="text/css">
  .default
  {
    font-size: 20px;
    text-align: left;
  }

  .red
  {
    color: red;
    font-size: 15px;
  }
</style>
```

In the following example, the `div` element uses the `default` style class above. Clicking the link adds the `red` style class to its style definitions:

```
<b:button b:action="addclass" b:value="red" b:target="//div">Click me!</b:button>
<div class="default">Click the button to change the font size and color.....</div>
```

### alert

Shows a modal JavaScript alert box with the given value as the content.

#### Attributes

**[Required] b:value**

A string containing a stated value.

### Implementation

Clicking on the `a` element results in a popup alert message displaying the text "Hi!".

```
<a b:action="alert" b:value="Hi!">Click here to show the alert box.</a>
```

### assign

Assigns a new value or nodelist (`b:select`) to a variable which is specified in the `b:target` attribute. To access a `tag` variable, you must targeting the element in which it is defined.

This command does not trigger a corresponding event.

#### Attributes

**[Required] b:target**

Specify an XPath expression pointing to the target element. The default value is `.` (self).

**[Required] b:select**

Specify the element that is selected using an XPath statement.

**b:scope**

Specify where the variable can be used:

A `local` variable (the default) is available within the scope of the element in which is triggered, and is passed automatically along with any event calls within the event handler. The variable is cleared from memory when the original execution thread ends.

A `global` variable is available throughout your application and exists for the lifespan of the application.

A `tag` variable is available within the scope of the element in which it is defined, and exists for the lifespan of the application.

**Values:**

`global`

`local`

`tag`

## Implementation

```
<div id="div">
    <s:event b:on="construct">
        <s:variable b:name="global" b:select="following-sibling::div" b:scope="global" />
        <s:variable b:name="tag" b:select="following-sibling::div" b:scope="tag" />
    </s:event>
    <s:event b:on="command">
        <s:task b:action="alert" b:value="{concat('You triggered a local var containing: ', $local)}" />
        <s:task b:action="assign" b:target="$local" b:select="id('div2')/@id" />
        <s:task b:action="alert" b:value="{concat('You changed the local var to : ', $local)}" />
    </s:event>
</div>
<div id="div1">Backbase
    <s:event b:on="command">
        <s:variable b:name="local" b:select="." b:scope="local" />
        <s:task b:action="trigger" b:event="command" b:target="id('div')" />
    </s:event>
    <div id="div2">Rich
        <div id="div3">Internet
            <div id="div4">Applications</div>
        </div>
    </div>
</div>
<a>Get and assign GLOBAL variable
    <s:event b:on="command">
        <s:task b:action="alert" b:value="${global}" />
        <s:task b:action="assign" b:target="$global" b:select="id('div2')" />
        <s:task b:action="alert" b:value="${global}" />
    </s:event>
</a>
|
<a>Get and assign TAG variable
    <s:event b:on="command">
        <s:task b:action="alert" b:value="{id('div')/$tag}" />
        <s:task b:action="assign" b:target="id('div')/$tag" b:select="'This is a TAG variable'" />
        <s:task b:action="alert" b:value="{id('div')/$tag}" />
    </s:event>
</a>
```

## backward

Instructs the system to go to the previous item in the history list. Specify the history list using the optional attribute `b:history`. If `b:history` is not specified, the global history list is taken. This command does not trigger a corresponding event.

**Attributes****b:history**

Specify the history thread to perform the action on.

**Implementation**

```
<s:behavior b:name="history" b:behavior="b-tab">
    <s:event b:on="select">
        <s:super />
        <s:history b:name="myHistory">
            <s:task b:action="select" />
        </s:history>
    </s:event>
</s:behavior>
<b:button b:action="backward" b:history="myHistory">&lt; Previous</b:button>
<b:button b:action="forward" b:history="myHistory">Next &gt;</b:button>
<b:tabbox style="height: 100%">
    <b:tab b:label="2001" b:behavior="history" />
    <b:tab b:label="Seven Samurai" b:behavior="history" />
    <b:tab b:label="Life is Beautiful" b:behavior="history" />
    <b:tab b:label="The Godfather" b:behavior="history" />
</b:tabbox>
```

**blur**

Removes focus from an input element.

You can use this command on elements in the *BXML Space* and *HTML Space*.

**Attributes****b:target**

Specify an XPath expression pointing to the target element. The default value is `.` (self).

**Implementation**

In the following example, when you leave an input element that has focus, the `blur` event is triggered. An event handler is defined in the behavior that changes the background color of an input element that no longer has focus.

```
<s:behavior b:name="focus-blur">
    <s:event b:on="blur">
        <s:setstyle b:background-color="#D6F8F8" />
    </s:event>
    <s:event b:on="focus">
        <s:setstyle b:background-color="white" />
    </s:event>
</s:behavior>
<div style="width: 300px; height: 200px; padding: 10px">
    <input b:behavior="focus-blur" id="focusitem" type="text" value="name" />
    <br />
    <br />
    <input b:behavior="focus-blur" type="text" value="age" />
    <br />
    <br />
    <input b:behavior="focus-blur" type="text" value="address" />
    <br />
    <br />
</div>
```

**bookmark**

Sets a bookmark with the name of the given value. This command does not trigger a corresponding event.

**Attributes****[Required] b:value**

A string containing a stated value.

## bufferdirty

Forces the buffer specified in the `b:target` attribute to reload its data source the next time it gets selected (used for the `b:buffer` element defined in a `b:deck`).

### Attributes

#### `b:target`

Specify an XPath expression pointing to the target element. The default value is `.` (self).

## Implementation

In the following example, a behavior is defined that has an event handler for the `select` event. When a `b:buffer` is selected, the `bufferdirty` action is triggered that forces the content of the file specified in the `b:url` to be reloaded.

```
<s:behavior b:name="bufferdirty">
    <s:event b:on="select">
        <s:task b:action="bufferdirty" b:target="." />
    </s:event>
</s:behavior>
<b:deck id="buffer_spideck" style="border: solid 1px #959595; width: 400px; height:400px">
    <b:buffer b:behavior="bufferdirty" b:url="data/seven.xml" />
    <b:buffer b:behavior="bufferdirty" b:url="data/2001.xml" />
    <b:buffer b:behavior="bufferdirty" b:url="data/bella.xml" />
</b:deck>
<b:button b:action="previous" b:target="id('buffer_spideck')">Previous</b:button>
<b:button b:action="next" b:target="id('buffer_spideck')">Next</b:button>
```

## bxml2string

Converts a BXML node-set to a String. If you specify only a `b:variable`, it uses it as the source and the target; it converts the BXML contained in the variable to a String and returns the result back into the variable.

### Attributes

#### [Required] `b:variable`

Specify the unique name of a variable, that has been declared using the `s:variable` tag, in which the result of the transformation is stored. The variable name must be prefixed by a \$ sign, for example `b:variable="$data"` where `data` is the variable name.

#### `b:source`

Specify the source element to use for the action using an XPath expression. The default is the current element.

Note you can also specify a variable for `b:source`, for example `b:source="$myvar"`.

#### `b:singletag`

Specify that only a single line is printed in the result (the default). Specify `false` to print the document from the specified source.

Values:  
false  
true

## Implementation

```
<implementation>
    <snippet try="yes" type="xml">
<b:box id="x">
    <div>My favourite film is the
        <span>Seven Sumarai</span>
    </div>
</b:box>
<b:button>Store BXML as String
<s:event b:on="command">
    <s:variable b:name="bxml" b:select="id('x')" b:scope="global" />
    <s:task b:action="alert" b:value="{$bxml}" />
```

```

        <s:task b:action="bxm2string" b:variable="$bxml" />
        <s:task b:action="alert" b:value="{$bxml}" />
    </s:event>
</b:button>
</snippet>
</implementation>
```

## copy

Copies an element, and its children, specified in the `b:source` to a different location in the BXML tree. Specify the receiving element using the the `b:destination` attribute. You can use this command on elements in the *BXML Space* and *HTML Space*.

The `copy` action triggers the events: `copy` and `receive`.

### Attributes

#### [Required] `b:source`

Specify the source element to use for the action using an XPath expression. The default is the current element.

Note you can also specify a variable for `b:source`, for example `b:source="$myvar"`.

#### [Required] `b:destination`

Specify an XPath expression pointing to the receiving/related element. The `b:destination` of a `load`, `xsl-transform`, `send`, or `submit` command can be a variable, for example `b:destination="$myvar"`, in which case the variable is resolved to a String, or it can target an element in the BXML tree, in which case it is resolved to BXML.

#### `b:mode`

Define where a node is inserted relative to the receiving element. By default, the value is set to `replacechildren`.

Values:

- after
- asfirstchild
- aslastchild
- before
- replace
- replacechildren

## Implementation

In the following example, you can click an item in the list to move it from one list to another:

```

<div id="container">
    <p>Click the articles to move them to your shopping list...</p>
    <b:box b:style="width: 400px; height: 100px">
        <ul id="products list">
            <li b:action="copy" b:source=". " b:destination="id('container')//ul except ..." b:mode="aslastchild">Cheese</li>
            <li b:action="copy" b:source=". " b:destination="id('container')//ul except ..." b:mode="aslastchild">Eggs</li>
            <li b:action="copy" b:source=". " b:destination="id('container')//ul except ..." b:mode="aslastchild">Milk</li>
            <li b:action="copy" b:source=". " b:destination="id('container')//ul except ..." b:mode="aslastchild">Bread</li>
            <li b:action="copy" b:source=". " b:destination="id('container')//ul except ..." b:mode="aslastchild">Bananas</li>
            <li b:action="copy" b:source=". " b:destination="id('container')//ul except ..." b:mode="aslastchild">Bacon</li>
        </ul>
    </b:box>
    <br />
    <b:box b:style="width: 400px; height: 150px">
        <p>My Shopping List:</p>
        <ul id="shopping list"></ul>
    </b:box>
</div>
```

## deselect

Brings the target element to its deselected state. The state is automatically reflected in everything that is dependent on the element which changed state (for example, behaviors and observing elements).

### Attributes

#### b:target

Specify an XPath expression pointing to the target element. The default value is . (self).

## Implementation

In the following example, the behavior contains three event handlers: when an element that uses the behavior is clicked, a `select` action is triggered that selects the current element, and a `deselect` action that deselects its sibling elements. The actions trigger the `select` event handler that shows the child `div` element of the element that is selected, and the `deselect` event handler that hides the child `div` element of all deselected elements.

```
<s:behavior b:name="open1child">
  <s:event b:on="command">
    <s:task b:action="select" b:target="." />
    <s:task b:action="deselect" b:target="following-sibling::div | preceding-sibling::div" />
  </s:event>
  <s:event b:on="select">
    <s:task b:action="show" b:target="div" />
  </s:event>
  <s:event b:on="deselect">
    <s:task b:action="hide" b:target="div" />
  </s:event>
</s:behavior>
<div>
  <div b:behavior="open1child" class="myClass">Seven Samurai</> &gt;
    <div style="font-size: 10px; display: none">A film by....</div>
  </div>
  <div b:behavior="open1child" class="myClass">Life is Beautiful</> &gt;
    <div style="font-size: 10px; display: none">A film by...</div>
  </div>
  <div b:behavior="open1child" class="myClass">Citizen Kane</> &gt;
    <div style="font-size: 10px; display: none">A film by</div>
  </div>
</div>
```

## disable

Disables an element so that it can no longer get focus by setting the `b:disabled` property to `true`. When an element is disabled, it cannot respond to user input events. You can enable an element using the `enable` command. The `disable` command triggers a `disable` event.

### Attributes

#### [Required] b:tasklist

Specify the name of the tasklist to execute. If multiple tasklists with the same name exist, the last defined tasklist will be executed.

## Implementation

In the following example, clicking the Yes button disables the input fields, clicking the No button enables the fields:

```
<div style="width: 300px; height: 130px; padding: 10px">
  <input type="text" value="name" />
```

```

<br />
<br />
<input type="text" value="age" />
<br />
<br />
<input type="text" value="address" b:behavior="enable-disable" />
<br />
<br />
    Are your details correct?
<b:button b:action="disable" b:target="//input">Yes</b:button>
<b:button b:action="enable" b:target="//input">No</b:button>

```

## enable

Enables an element to get focus and respond to user input events by setting the `b:disabled` property to `false`. The `enable` command triggers an `enable` event.

### Attributes

**[Required] `b:tasklist`**

Specify the name of the tasklist to execute. If multiple tasklists with the same name exist, the last defined tasklist will be executed.

### Implementation

In the following example, clicking the Yes button disables the input fields, clicking the No button enables the fields:

```

<div style="width: 300px; height: 130px; padding: 10px">
    <input type="text" value="name" />
    <br />
    <br />
    <input type="text" value="age" />
    <br />
    <br />
    <input type="text" value="address" b:behavior="enable-disable" />
    <br />
    <br />
    Are your details correct?
    <b:button b:action="disable" b:target="//input">Yes</b:button>
    <b:button b:action="enable" b:target="//input">No</b:button>

```

## execute

Runs the specified `tasklist`. This command does not trigger a corresponding event.

### Attributes

**[Required] `b:tasklist`**

Specify the name of the tasklist to execute. If multiple tasklists with the same name exist, the last defined tasklist will be executed.

### Implementation

In the following example, clicking the link triggers the `execute` command that executes the `s:tasklist` called `mytasklist`:

```

<a b:action="execute" b:tasklist="mytasklist">Click here to run the tasklist.</a>
<s:tasklist b:name="mytasklist">
    <s:task b:action="alert" b:value="Hello!" />
    <s:task b:action="set" b:target="//a/@style" b:value="color: red; font-size: larger" />
</s:tasklist>

```

## focus

Puts focus on a focusable element (an input element, or a `b:focusitem` element).

You can use this command on elements in the *BXML Space* and *HTML Space*.

## Attributes

### b:target

Specify an XPath expression pointing to the target element. The default value is . (self).

## Implementation

In the following example,

```
<s:event b:on="construct">
  <s:task b:action="focus" b:target="id('focusitem')"/>
</s:event>
<div style="width: 300px; height: 200px; padding: 10px">
  <input id="focusitem" type="text" value="name" />
  <br />
  <br />
  <input type="text" value="age" />
  <br />
  <br />
  <input type="text" value="address" />
  <br />
  <br />
  <textarea value="add comments" />
</div>
```

## focusgroup-next

Navigates to the next focusgroup element in the hierarchy. This command does not trigger a corresponding event.

## Implementation

In the following example, the textareas use a behavior in which keys are defined that overrule the default tab and tab+shift keys used to navigate between focusgroups (input elements are by default focusgroups and focusitems). Use the right and left arrow keys to go to the next/previous focusgroup:

```
<s:event b:on="construct">
  <s:task b:action="focus" b:target="//textarea" />
</s:event>
<s:behavior b:name="focus1" b:focusitem="true" b:focusindicator="true">
  <s:whenactive>
    <s:keys b:keys="right" b:action="focusgroup-next" b:bubble="true" />
    <s:keys b:keys="left" b:action="focusgroup-previous" b:bubble="true" />
    <s:keys b:keys="tab" b:bubble="false" />
    <s:keys b:keys="tab+shift" b:bubble="false" />
  </s:whenactive>
</s:behavior>
<div style="position: absolute; left: 50px; top: 50px;">
  <textarea b:behavior="focus1">Textarea 1</textarea>
  <textarea b:behavior="focus1">Textarea 2</textarea>
  <textarea b:behavior="focus1">Textarea 3</textarea>
</div>
```

## focusgroup-previous

Navigates to the previous focusgroup element in the hierarchy. This command does not trigger a corresponding event.

## Implementation

In the following example, the textareas use a behavior in which keys are defined that overrule the default tab and tab+shift keys used to navigate between focusgroups (input elements are by default focusgroups and focusitems). Use the right and left arrow keys to go to the next/previous focusgroup:

```
<s:event b:on="construct">
```

```

<s:task b:action="focus" b:target="//textarea" />
</s:event>
<s:behavior b:name="focus1" b:focusitem="true" b:focusindicator="true">
    <s:whenactive>
        <s:keys b:keys="right" b:action="focusgroup-next" b:bubble="true" />
        <s:keys b:keys="left" b:action="focusgroup-previous" b:bubble="true" />
        <s:keys b:keys="tab" b:bubble="false" />
        <s:keys b:keys="tab+shift" b:bubble="false" />
    </s:whenactive>
</s:behavior>
<div style="position: absolute;left:50px; top:50px;">
    <textarea b:behavior="focus1">Textarea 1</textarea>
    <textarea b:behavior="focus1">Textarea 2</textarea>
    <textarea b:behavior="focus1">Textarea 3</textarea>
</div>

```

## focusitem-next

Puts focus on the next focusitem element in the hierarchy. This command does not trigger a corresponding event.

### Implementation

In the following example,

```

<s:behavior b:name="focus1">
    <s:initatt b:focusitem="true" b:focusindicator="true" />
    <s:whenactive>
        <s:keys b:keys="a+s" b:action="focusitem-next" b:bubble="true" />
        <s:keys b:keys="q+w" b:action="focusitem-previous" b:bubble="true" />
        <s:keys b:keys="right down left up" b:bubble="false" />
    </s:whenactive>
</s:behavior>
<div focusgroup="true" style="position: absolute;left:50px; top:50px; width: 250px; height: 200px; border: solid 1px">
    <div>Press a+s to go to the next item..</div>
    <div>Press q+w to go to the previous item..</div>
    <ul>
        <li b:behavior="focus1">Apples</li>
        <li b:behavior="focus1">Pears</li>
        <li b:behavior="focus1">Bananas</li>
        <li b:behavior="focus1">Grapes</li>
        <li b:behavior="focus1">Peaches</li>
    </ul>
</div>

```

## focusitem-previous

Puts focus on the previous focusitem element in the hierarchy. This command does not trigger a corresponding event.

### Implementation

In the following example,

```

<s:behavior b:name="focus1">
    <s:initatt b:focusitem="true" b:focusindicator="true" />
    <s:whenactive>
        <s:keys b:keys="a+s" b:action="focusitem-next" b:bubble="true" />
        <s:keys b:keys="q+w" b:action="focusitem-previous" b:bubble="true" />
        <s:keys b:keys="right down left up" b:bubble="false" />
    </s:whenactive>
</s:behavior>
<div focusgroup="true" style="position: absolute;left:50px; top:50px; width: 250px; height: 200px; border: solid 1px">
    <div>Press a+s to go to the next item..</div>
    <div>Press q+w to go to the previous item..</div>
    <ul>
        <li b:behavior="focus1">Apples</li>
        <li b:behavior="focus1">Pears</li>
        <li b:behavior="focus1">Bananas</li>

```

```

<li b:behavior="focus1">Grapes</li>
<li b:behavior="focus1">Peaches</li>
</ul>
</div>

```

## focusroot-next

Navigates to the next focusroot element in the hierarchy. This command does not trigger a corresponding event.

## focusroot-previous

Navigates to the previous focusroot element in the hierarchy. This command does not trigger a corresponding event.

## formxml

Converts the contents of a `form` element to basic XML. The `b:source` must target an element in which a `form` element is defined. The contents of the form is converted to a generic XML document which is inserted into the variable indicated by the `b:variable`.

### Attributes

#### [Required] `b:source`

Specify the source element to use for the action using an XPath expression. The default is the current element.

Note you can also specify a variable for `b:source`, for example `b:source="$myvar"`.

#### [Required] `b:variable`

Specify the unique name of a variable, that has been declared using the `s:variable` tag, in which the result of the transformation is stored. The variable name must be prefixed by a \$ sign, for example `b:variable="$data"` where `data` is the variable name.

## Implementation

In the following example, the `formxml` command converts the contents of the `form` element to XML and returns the result in a variable `$formdata`, which is then displayed in an alert box:

```

<div id="f1">
  <form name="form1">
    <span b:formelement="true" b:name="xxx" b:value="whatever">Enter Your Personal
    Details</span>
    <div>
      <div>
        <div>
          <input name="firstname" value="John" xxx="xx" group="1" />
          <input name="lastname" value="Doe" xxx="xx" group="1" />
        </div>
      </div>
    </div>
  </form>
  <form name="form2">
    <span b:value="whatever">Add your remarks:</span>
    <div>
      <textarea>Comments...</textarea>
    </div>
  </form>
</div>
<a>Transform the form data to XML:
  <s:event b:on="command">
    <s:variable b:name="formdata" />
    <s:task b:action="formxml" b:variable="$formdata" b:source="id('f1')" />

```

```

<s:task b:action="alert" b:value="{$formdata}" />
</s:event>
</a>
```

## Results in the following XML:

```

<forms id="f1" xmlns="http://www.w3.org/1999/xhtml" xmlns:b="http://www.backbase.com/b"
xmlns:s="http://www.backbase.com/s">
<form name="form1">
    <span b:formelement="true" b:name="xxx" b:value="whatever">whatever</span>
    <input name="firstname" value="John" xxx="xx" group="1">John</input>
    <input name="lastname" value="Doe" xxx="xx" group="1">Doe</input>
</form>
<form name="form2">
    <textarea value="Comments...">Comments...</textarea>
</form>
</forms>
```

## forward

Instructs the system to go to the next item in the history list. Specify the history list using the optional attribute **b:history**. If **b:history** is not specified, the global history list is taken. This command does not trigger a corresponding event.

### Attributes

#### **b:history**

Specify the history thread to perform the action on.

## Implementation

```

<s:behavior b:name="history" b:behavior="b-tab">
    <s:event b:on="select">
        <s:super />
        <s:history b:name="myHistory">
            <s:task b:action="select" />
        </s:history>
    </s:event>
</s:behavior>
<b:button b:action="backward" b:history="myHistory">&lt; Previous</b:button>
<b:button b:action="forward" b:history="myHistory">Next &gt;</b:button>
<b:tabbox style="height: 100%">
    <b:tab b:label="2001" b:behavior="history" />
    <b:tab b:label="Seven Samurai" b:behavior="history" />
    <b:tab b:label="Life is Beautiful" b:behavior="history" />
    <b:tab b:label="The Godfather" b:behavior="history" />
</b:tabbox>
```

## fxtile

Tiles and animates the elements to their positions (similar to a [tile](#) command, but with animation effect). The targeted elements need to be positioned absolutely.

When you use the [fxtile](#) command, ensure that ancestor elements do not have the `style="display:none"` property set. The command can experience problems with drawing its target element if this property is set.

### Attributes

#### [Required] **b:target**

Specify an XPath expression pointing to the target element. The default value is `.` (self).

#### **b:orientation**

Specify the orientation of elements displayed in a tile formation. The default is `cols`.

#### Values:

`cols`

`rows`

## Implementation

The following example uses the `fxtile` command to tile a series of blocks displaying the text BACKBASE; click the links to arrange these blocks vertically or horizontally:

```
<style type="text/css">      .logoletter {      width: 20px;      height: 10px;
position: absolute;      font-size: 22px;      line-height: 22px;      font-weight: bold;
color: #C00;      border: 1px solid #CCC;      } </style>
<s:behavior b:name="tile">
  <s:event b:on="select">
    <s:task b:action="fxtile" b:target="div" b:orientation="cols" />
  </s:event>
  <s:event b:on="deselect">
    <s:task b:action="fxtile" b:target="div" b:orientation="rows" />
  </s:event>
</s:behavior>
<p>
  <a b:action="select" b:target="id('container')">Tile "Backbase" by column</a>
  <span>|</span>
  <a b:action="deselect" b:target="id('container')">Tile "Backbase" by row</a>
</p>
<div style="position: relative; height: 200px;" b:behavior="tile" id="container">
  <div id="div1" class="logoletter">B</div>
  <div id="div2" class="logoletter">A</div>
  <div id="div3" class="logoletter">C</div>
  <div id="div4" class="logoletter">K</div>
  <div id="div5" class="logoletter">B</div>
  <div id="div6" class="logoletter">A</div>
  <div id="div7" class="logoletter">S</div>
  <div id="div8" class="logoletter">E</div>
</div>
```

## hide

Hides the target element by setting its `display` style property to `none`.

You can use this command on elements in the *BXML Space* and *HTML Space*.

### Attributes

#### b:target

Specify an XPath expression pointing to the target element. The default value is `.` (self).

## Implementation

In the following example, clicking the link hides the `span` element by setting the style to `display:none`.

```
<a b:action="hide" b:target=".../p[span]">Click here to hide the text below.</a>
<p>
  <span>If you don't want to display this element, click the link above...</span>
</p>
<p>Note that the position of this element is moved up....</p>
```

## html2string

Converts HTML to a String. If you specify only a `b:variable`, it uses it as the source and the target; it converts the HTML contained in the variable to a String and returns the result back into the variable.

### Attributes

#### [Required] b:variable

Specify the unique name of a variable, that has been declared using the `s:variable` tag, in which the result of the transformation is stored. The variable name must be prefixed by a \$ sign, for example `b:variable="$data"` where `data` is the variable name.

#### b:source

Specify the source element to use for the action using an XPath expression.

The default is the current element.

Note you can also specify a variable for `b:source`, for example `b:source="$myvar"`.

## Implementation

In the following example, the `html2string` command converts the BXML contained in a variable `$html` to a String:

```
<s:task b:action="html2string" b:variable="$html" />
```

## invisible

Sets the `visibility` style property of the target element to `hidden`.

You can use this command on elements in the *BXML Space* and *HTML Space*.

### Attributes

#### `b:target`

Specify an XPath expression pointing to the target element. The default value is `.` (self).

## Implementation

```
<a b:action="invisible" b:target="../b:box[1]">Click here to make the element below
invisible</a>
<b:box style="width: 150px">When you click the link, this box disappears...</b:box>
<b:box style="width: 150px">The position of this element remains the same
regardless....</b:box>
```

## js

Evaluates the fragment of JavaScript code specified in the `b:value` attribute. You can also use Backbase JavaScript functions in your code. For more information, see the section *JavaScript functions*.

### Attributes

#### [Required] `b:value`

The JavaScript code block to be evaluated. Note that the `&` and `<` characters have to be encoded as `&amp;` and `&lt;`.

## Implementation

In the following example, clicking the link displays a pop-up alert message showing the date and time the document was last modified:

```
<a b:action="js" b:value="alert(document.lastModified)">Click here to show the modification
date of this document.</a>
```

## load

Loads data from an external file, which is parsed by the BPC engine and inserted in the document at the specified destination. (The `s:httpheader` tag allows you to create custom HTTP header information.)

This command does not trigger a corresponding event.

The `load` command does not work across domains. To call, for example, a web service from a Backbase application (that does not reside on the same server as the application), you have to write your own server-side proxy.

### Attributes

#### [Required] `b:url`

Specify the path of the file to load, relative to the startup file of your Backbase application.

**b:destination**

Specify an XPath expression pointing to the receiving/related element. The **b:destination** of a `load`, `xsl-transform`, `send`, or `submit` command can be a variable, for example `b:destination="$myvar"`, in which case the variable is resolved to a String, or it can target an element in the BXML tree, in which case it is resolved to BXML.

**b:mode**

Define where a node is inserted relative to the receiving element. By default, the value is set to `replacechildren`.

Values:  
 after  
 asfirstchild  
 aslastchild  
 before  
 replace  
 replacechildren

**b:method**

Specify the method used to load data.

Values:  
 custom  
 get  
 post  
 xml

**b:data**

Specify the String data. If the data is stored in a variable, you can get the data using the syntax: `b:data="{$varname}"` (curly brackets indicate that the contents is an XPath expression that must be resolved).

## Implementation

In the following example, when you click the `div` element, the contents of the loaded file are inserted

```
<s:behavior b:name="load">
  <s:event b:on="command">
    <s:task b:action="load" b:url="sumarai.xml" b:destination="." b:mode="replace" />
  </s:event>
</s:behavior>
<div b:behavior="load">Load contents here</div>
```

Where the contents of the loaded file (`sumarai.xml`) are:

```
<?xml version="1.0" ?>
<div xmlns="http://www.w3.org/1999/xhtml" xmlns:b="http://www.backbase.com/b"
      xmlns:s="http://www.backbase.com/s">
  <h3>The Seven Samurai (1954)</h3>
  <p>Directed by Akira Kurosawa and starring Takashi Shimura and Toshiro Mifune, the film takes place in war-ridden 16th century Japan, where a village of farmers look for ways to ward off a band of marauding robbers.</p>
</div>
```

## maximize

Resizes an element to the full width and height of its containing element.

You can use this command on elements in the *BXML Space* and *HTML Space*.

### Attributes

**b:target**

Specify an XPath expression pointing to the target element. The default value is `.` (self).

## Implementation

In the following example, the resizing and restoring of the element is defined within a behaviour; click the links to maximize and restore:

```
<s:behavior b:name="max">
```

```

<s:event b:on="select">
    <s:task b:action="maximize" b:target="." />
</s:event>
<s:event b:on="deselect">
    <s:task b:action="restore" b:target="." />
</s:event>
</s:behavior>
<p style="position: relative;">
    <a b:action="select" b:target="id('res')">Maximize</a>
    <span>|</span>
    <a b:action="deselect" b:target="id('res')">Restore</a>
</p>
<div style="position: relative; border: 1px solid #888; width: 300px; height: 200px;
color:white; background: green">
    <div style="position: absolute; background: red;" b:behavior="max" id="res">
        <p>Maximized and Restored element!</p>
    </div>
</div>

```

## move

Moves an element, and its children, specified in the `b:source` to a different location in the BXML tree. Specify the receiving element using the the `b:destination` attribute. You can use this command on elements in the *BXML Space* and *HTML Space*.

The `move` action triggers the events: `move`, `receive`, and `child-lost`.

### Attributes

#### `b:source`

Specify the source element to use for the action using an XPath expression. The default is the current element.

Note you can also specify a variable for `b:source`, for example `b:source="$myvar"`.

#### [Required] `b:destination`

Specify an XPath expression pointing to the receiving/related element. The `b:destination` of a `load`, `xsl-transform`, `send`, or `submit` command can be a variable, for example `b:destination="$myvar"`, in which case the variable is resolved to a String, or it can target an element in the BXML tree, in which case it is resolved to BXML.

#### `b:mode`

Define where a node is inserted relative to the receiving element. By default, the value is set to `replacechildren`.

##### Values:

- after
- asfirstchild
- aslastchild
- before
- replace
- replacechildren

## Implementation

In the following example, you can click an item in the list to move it from one list to another:

```

<div id="container">
    <p>Click the articles to move them to your shopping list...</p>
    <b:box b:style="width: 400px; height: 100px">
        <ul id="products list">
            <li b:action="move" b:source=".." b:destination="id('container')//ul except ..." b:mode="aslastchild">Cheese</li>
            <li b:action="move" b:source=".." b:destination="id('container')//ul except ..." b:mode="aslastchild">Eggs</li>
            <li b:action="move" b:source=".." b:destination="id('container')//ul except ..." b:mode="aslastchild">Milk</li>
            <li b:action="move" b:source=".." b:destination="id('container')//ul except ..." b:mode="aslastchild">Bread</li>
            <li b:action="move" b:source=".." b:destination="id('container')//ul except ..." b:mode="aslastchild">Bacon</li>
        </ul>
    </b:box>

```

```

b:mode="aslastchild">Bananas</li>
    <li b:action="move" b:source=". " b:destination="id('container')//ul except .."
b:mode="aslastchild">Bacon</li>
</ul>
</b:box>
<br />
<b:box b:style="width: 400px; height: 150px">
    <p>My Shopping List:</p>
    <ul id="shopping list"></ul>
</b:box>
</div>

```

## movedown

Causes the target element to switch positions with its next sibling, effectively moving the element down one in the list.

### Attributes

#### **b:target**

Specify an XPath expression pointing to the target element. The default value is . (self).

## Implementation

In the following example, when you click a item in the list it moves down in the list order.

```

<p>What sort of films do you like most?</p>
<ol>
    <li>
        <a b:action="movedown" b:target=". .">Drama</a>
    </li>
    <li>
        <a b:action="movedown" b:target=". .">Thriller</a>
    </li>
    <li>
        <a b:action="movedown" b:target=". .">Comedy</a>
    </li>
    <li>
        <a b:action="movedown" b:target=". .">Horror</a>
    </li>
    <li>
        <a b:action="movedown" b:target=". .">Musical</a>
    </li>
</ol>

```

## moveup

Causes the target element to switch positions with its previous sibling, effectively moving the element up one in the list.

### Attributes

#### **b:target**

Specify an XPath expression pointing to the target element. The default value is . (self).

## Implementation

In the following example, when you click a item in the list it moves up one in the list order.

```

<p>What sort of films do you like the most?</p>
<ol>
    <li b:action="moveup" b:target=". ">Drama</li>
    <li b:action="moveup" b:target=". ">Thriller</li>
    <li b:action="moveup" b:target=". ">Comedy</li>
    <li b:action="moveup" b:target=". ">Horror</li>
    <li b:action="moveup" b:target=". ">Musical</li>
</ol>

```

## msg

Define a custom message. Messages are generated in the Runtime Tracer of the Backbase Developer Tool.

### Attributes

#### [Required] b:value

A string containing a stated value.

#### b:level

Specify the messaging level. The default is `custom`.

##### Values:

- `critical`
- `custom`
- `debug`
- `error`
- `info`
- `warning`

## Implementation

```
<a>Click here to generate a custom message...
  <s:event b:on="command">
    <s:task b:action="msg" b:value="...generating a custom message" />
  </s:event>
</a>
```

## next

Selects the next sibling of the currently selected child of the targeted element. The command is used mostly in the `b:deck` control.

### Attributes

#### b:target

Specify an XPath expression pointing to the target element. The default value is `.` (self).

## Implementation

In the following example, clicking the Next link selects the next child element in the `b:deck` control:

```
<style> .b-deck { height: 200px; width: 400px; overflow: auto; border: 3px double #CCD; padding: .5em; }</style>
<b:deck class="b-deck">
  <div>1. The Seven Samurai was made in 1954...</div>
  <div>2. It was directed by Akira Kurosawa and stars Takashi Shimura and Toshiro Mifune...</div>
  <div>3. The film takes place in war-ridden 16th century Japan, where a village of farmers look for ways to ward off a band of marauding robbers.</div>
</b:deck>
<a b:action="previous" b:target="..></b:deck">&lt; Previous</a>
<a b:action="next" b:target="..></b:deck">Next &gt;</a>
```

## opacity

Make an element opaque by specifying a `b:value` from between `0` and `100`, where `10` reduces the element's shading to 10% of its original luminance.

You can use this command on elements in the *BXML Space* and *HTML Space*.

### Attributes

#### [Required] b:value

A string containing a stated value.

## Implementation

In the following example, clicking a link triggers an event handler defined within a behavior that sets the opacity to 50%:

```

<s:behavior b:name="opaque">
  <s:event b:on="command">
    <s:task b:action="opacity" b:value="50" />
  </s:event>
</s:behavior>
<ul>
  <li>
    <a b:behavior="opaque">Seven Sumari</a>
  </li>
  <li>
    <a b:behavior="opaque">Citizen Kane</a>
  </li>
  <li>
    <a b:behavior="opaque">The African Queen</a>
  </li>
</ul>

```

## position

Changes the position of an element on the screen. Depending on the use of the **b:type** attribute, there are four principle means of positioning an element on the screen. Therefore when you decide to use the **position** command you must first determine which type of positioning command you wish to perform and then select the appropriate value for the **b:type** attribute. Available types of positioning commands:

- **Absolute** Positioning Command  
positions the target element absolutely. The new position is determined by one or by a combination of the values of the **b:top**, **b:bottom**, **b:left** and **b:right** attributes. Depending on how they are combined, they may not only affect the new position of the element but also the height and the width.
- **Shift** Positioning Command  
shifts the position of the target element relatively to its current position. One or more of the **b:top**, **b:bottom**, **b:left** and **b:right** attributes are used to determine the new position. Two opposite attributes such as **b:left** and **b:right** may not be used together.
- **Align** Positioning Command  
aligns the target element to its containing box. This **b:placement** attribute determines how the target element is aligned.
- **Place** Positioning Command  
repositions the target element relatively to another element. This second element is specified by the **b:destination** attribute. How exactly the target element is positioned relative to the anchor element is determined by the **b:position** attribute.

You can use this command on elements in the **BXML Space** and **HTML Space**.

### Attributes

#### [Required] **b:type**

Define the type of positioning used by the **position** command. The type also determines which attributes are required and/or available.

Values:  
absolute  
align  
place  
shift

#### **b:target**

Specify an XPath expression pointing to the target element. The default value is `.` (self).

#### **b:left**

Specify a percentage or pixel value that sets the left position from the left

edge of the containing element. Negative values are allowed.

#### b:top

Specify a percentage or pixel value that sets the top position from the top edge of the containing element. Negative values are allowed.

#### b:bottom

A percentage or pixel value that sets the bottom position from the bottom edge of the containing element. Negative values are allowed.

#### b:right

Specify a percentage or pixel value that sets the right position from the right edge of the containing element. Negative values are allowed.

#### b:placement

Specify the alignment value that sets the position of an element relative to another element.

Values:

center  
left  
right

#### b:destination

Specify an XPath expression pointing to the receiving/related element. The b:destination of a `load`, `xsl-transform`, `send`, or `submit` command can be a variable, for example `b:destination="$myvar"`, in which case the variable is resolved to a String, or it can target an element in the BXML tree, in which case it is resolved to BXML.

#### b:position

The alignment value that sets the position of a popup element relative to a target element. The first word represents the side of the target which should be adjacent to the opposite side of the popup element and the second word represents the sides of the target and popup elements which should be aligned as if parallel to each other:

- `before` maps to the `top` side.
- `after` maps to the `bottom` side.
- `start` maps to the `left` side.
- `end` maps to the `right` side.

Values:

after-end  
after-pointer  
after-start  
at-pointer  
before-end  
before-start  
end-after  
end-before  
overlap  
start-after  
start-before

## Implementation

In the following example, the `position` command of type `place` is used; when you left-click the `div` element, text is displayed and positioned at the mouse pointer. When you right-click the `div` element, text is displayed and is positioned in the same vertical plane as where the mouse pointer was clicked, but after the bottom edge. Each time you left-click or right-click, the positioning of the pop-up text is redefined according to the mouse pointer:

```
<div id="place" style="position: relative; border: 1px solid #CE0000; width: 250px; height: 100px;">
  <s:event b:on="click" b:action="position" b:type="place" b:target="id('target1')"
  b:destination="id('place')" b:position="at-pointer" />
  <s:event b:on="click" b:action="show" b:target="id('target1')" />
  <s:event b:on="rmbdown" b:action="position" b:type="place" b:target="id('target2')"
  b:destination="id('place')" b:position="after-pointer" b:bubble="false" />
  <s:event b:on="rmbdown" b:action="show" b:target="id('target2')" b:bubble="false" />
  Try left-clicking in the box...
</div>
```

```

<br />
    Try right-clicking in the box...
</div>
<div id="target1" style="display: none; position: absolute; background: #DEDEEF;">      ...to
display this text where you clicked with your mouse</div>
<div id="target2" style="display: none; position: absolute; background: #DCE3E7;">      ...to
display this text under the bottom edge of
<br />
    the area, aligned to where you clicked with your mouse
</div>

```

## previous

Selects the previous sibling of the currently selected child of the targeted element. The command is used mostly in the `b:deck` control.

### Attributes

#### `b:target`

Specify an XPath expression pointing to the target element. The default value is `.` (self).

### Implementation

In the following example, clicking the Previous link selects the previous child element in the `b:deck` control:

```

<style>    .b-deck {        height: 200px;        width: 400px;        overflow: auto;
border: 3px double #CCD;        padding: .5em;    }</style>
<b:deck class="b-deck">
    <div>1. The Seven Samurai was made in 1954...</div>
    <div>2. It was directed by Akira Kurosawa and stars Takashi Shimura and Toshiro
Mifune...</div>
    <div>3. The film takes place in war-ridden 16th century Japan, where a village of farmers
look for ways to ward off a band of marauding robbers.</div>
</b:deck>
<a b:action="previous" b:target=".//b:deck">&lt; Previous</a>
<a b:action="next" b:target=".//b:deck">Next &gt;</a>

```

## print

Opens the browser print window.

### Implementation

In the following example,

```

<h3>The Seven Samurai (1954)</h3>
<p>Directed by Akira Kurosawa and starring Takashi Shimura and Toshiro Mifune, the film takes
place      in war-ridden 16th century Japan, where a village of farmers look for ways to ward
off a band of marauding robbers.</p>
<b:button b:action="print">Print</b:button>

```

## remove

Removes the target element, and its children, from the document tree. You can target a node-set, attribute nodes, or text nodes.

You can use this command on elements in the *BXML Space* and *HTML Space*.

### Attributes

#### `b:target`

Specify an XPath expression pointing to the target element. The default value is `.` (self).

### Implementation

In the following example, clicking the first link removes the sibling `p` element (consequently, all elements below it are moved up). Clicking the link again removes the

next `p` element. Clicking the second element removes the value attribute from the input type.

```
<a b:action="remove" b:target="..//p[1]">Click here to remove the paragraph</a>
<span>|</span>
<a b:action="remove" b:target="..//div/input/@value">Click here to remove the input
value</a>
<p>This paragraph will be removed.</p>
<p>Click again to remove this paragraph as well.</p>
<div>
    <input type="text" value="John Doe" />
</div>
```

## removeclass

Removes the style class, specified in the `b:value` attribute, from the style classes used by the target element.

You can use this command on elements in the *BXML Space* and *HTML Space*.

### Attributes

#### [Required] `b:value`

A string containing a stated value.

#### `b:target`

Specify an XPath expression pointing to the target element. The default value is `.` (self).

## Implementation

```
<style type="text/css">      .text {      color: yellow;      font-size: 20px;      }
.background {      background-color: green      }</style>
<b:box style="width: 300px">
    <b:button b:action="removeclass" b:value="background" b:target="..//p">Remove Background
Color</b:button>
    <p b:action="removeclass" b:value="text" class="text background">Hello World!</p>
</b:box>
```

## render

Executes a `s:render` tag by name. You can define the `b:destination` and `b:mode` attributes in the `s:render` tag, or in the `render` command.

The `render` action triggers a `receive` event.

### Attributes

#### [Required] `b:render`

Specify the name identifying the `s:render` tag in which the content to be rendered is defined.

#### `b:destination`

Specify an XPath expression pointing to the receiving/related element. The `b:destination` of a `load`, `xsl-transform`, `send`, or `submit` command can be a variable, for example `b:destination="$myvar"`, in which case the variable is resolved to a String, or it can target an element in the BXML tree, in which case it is resolved to BXML.

#### `b:context`

Specify the context in which the task is executed. The attribute associates the task (`s:execute` or `render` command) with a BXML element. The value is an XPath expression, for example `b:context="/"` defines the root element. By default, the context is the parent of the task element.

#### `b:mode`

Define where a node is inserted relative to the receiving element. By default, the value is set to `replacechildren`.

##### Values:

after  
asfirstchild

```
aslastchild
before
replace
replacechildren
```

## Implementation

In the following example, clicking the link triggers the `render` command:

```
<s:render b:name="render">
  <div>Item</div>
</s:render>
<a b:action="render" b:render="render" b:destination="id('copyTarget')"
b:mode="aslastchild">Copy "item" to the box.</a>
<b:box style="width: 100px; height: 200px">
  <div id="copyTarget"></div>
</b:box>
<a b:action="remove" b:target="id('copyTarget')/*">Clear copied items</a>
```

## resize

Resizes a targeted element.

### Attributes

#### b:target

Specify an XPath expression pointing to the target element. The default value is `.` (self).

#### b:width

Specify a percentage or pixel value specifying the width of an element.

#### b:height

Specify a percentage or pixel value for the height of an element.

#### b:constrain

If set to `true`, it restricts an element that can be resized from being able to go outside its bounding box.

#### b:type

Defines whether resize is absolute (for pixel values) or proportional (for percent values).

##### Values:

absolute

proportional

## Implementation

Click on the element to resize it to 150x60.

```
<div style="margin: 5px; border: 1px solid #000; background: #009AFE; color: white">
  <p b:action="resize" b:target=".." b:width="20%" b:height="80%">
    b:type="proportional">Click here to resize this box!</p>
</div>
<br />
<div style="margin: 5px; border: 1px solid #000; background: green; color: white">
  <p b:action="resize" b:target=".." b:width="150px" b:height="250px">
    b:type="absolute">Click here to resize this box!</p>
</div>
```

## restore

Resets the size of an element which has received a `maximize` or `store` command.

### Attributes

#### b:target

Specify an XPath expression pointing to the target element. The default value is `.` (self).

## Implementation

In the following example, the resizing and restoring of the element is defined within a behaviour; click the links to maximize and restore:

```

<s:behavior b:name="max">
    <s:event b:on="select">
        <s:task b:action="maximize" b:target=". " />
    </s:event>
    <s:event b:on="deselect">
        <s:task b:action="restore" b:target=". " />
    </s:event>
</s:behavior>
<p style="position: relative;">
    <a b:action="select" b:target="id('res')">Maximize</a>
    <span>|</span>
    <a b:action="deselect" b:target="id('res')">Restore</a>
</p>
<div style="position: relative; border: 1px solid #888; width: 300px; height: 200px; color:white; background: green">
    <div style="position: absolute; background: red; b:behavior="max" id="res">
        <p>Maximized and Restored element!</p>
    </div>
</div>

```

## resync

Resets the relationship an element has with another element defined using the **b:followstate** attribute. By default, when an element follows the state of another, the element remains attached to that element even if the original XPath, that was used to attach it to the element, is now pointing to a different element. The **resync** command forces a new lookup of the XPath.

This command does not trigger a corresponding event.

You can use this command on elements in the **BXML Space** and **HTML Space**.

### Attributes

#### [Required] **b:followstate**

Specify an XPath expression pointing to the element whose state is followed by the **b:followstate** element.

#### **b:target**

Specify an XPath expression pointing to the target element. The default value is **.** (self).

## Implementation

In the following example, the **div** elements follow the state of Link2; clicking the link selects/deselects the link, resulting in the child **div** elements opening or closing. When you drag-and-drop the link, the followstate relationship is reset; the **drag-receive** event handler has a **resync** action that makes the Link2 element the element that is followed.

```

<style type="text/css">.myClass {font-size: 16px; line-height: 22px; border: 1px solid #CCC; width: 200px }</style>
<s:behavior b:name="openChildren">
    <s:event b:on="select">
        <s:task b:action="show" b:target="div" />
    </s:event>
    <s:event b:on="deselect">
        <s:task b:action="hide" b:target="div" />
    </s:event>
</s:behavior>
<s:behavior b:name="follow">
    <s:event b:on="command">
        <s:task b:action="select-deselect" b:target=". " />
    </s:event>
</s:behavior>
<div id="x">
    <a b:behavior="follow" id="link2">Link 2 (followstate after resync)</a>
    <a b:behavior="follow" b:drag="x" id="link1">Link1 1 (followstate before drag)</a>
    <div b:behavior="openChildren" b:followstate="preceding-sibling::a[1]" class="myClass">Seven Samurai&gt;&gt;
        <div style="font-size: 10px; display: none">A film by....</div>

```

```

</div>
<div b:behavior="openChildren" b:followstate="preceding-sibling::a[1]"
class="myClass">Life is Beautiful&gt;&gt;
    <div style="font-size: 10px; display: none">A film by...</div>
</div>
<div b:dragreceive="x" style="border: 1px solid red; width: 300px; height: 100px; position:
absolute; top: 20px; left: 300px; padding: 10px">
    <s:event b:on="drag-receive">
        <s:task b:action="resync" b:followstate="id('link2')" b:target="id('x')/div" />
    </s:event>
    Drag-and-drop link2 here
</div>

```

## scrollto

Causes the containing elements with the CSS style property `overflow` set to `auto` or `scroll` to scroll to the specified targeted element. This also applies to nested container elements.

You can use this command on elements in the *BXML Space* and *HTML Space*.

### Implementation

```

<div style="overflow:scroll; height: 300px; width: 200px; border: solid 1px">
    <a b:action="scrollto" b:target="following-sibling::a[1]" b:animate="true">Go to end</a>
    <h1>Heading 1</h1>
    <p>Content...</p>
    <h1>Heading 2</h1>
    <p>Content...</p>
    <h1>Heading 3</h1>
    <p>Content...</p>
    <h1>Heading 4</h1>
    <p>Content...</p>
    <h1>Heading 5</h1>
    <p>Content...</p>
    <h1>Heading 6</h1>
    <p>Content...</p>
    <h1>Heading 7</h1>
    <p>Content...</p>
    <a b:action="scrollto" b:target="preceding-sibling::a[1]" b:animate="true">Go to
start</a>
</div>

```

#### Attributes

##### **b:target**

Specify an XPath expression pointing to the target element. The default value is `.` (self).

##### **b:animate**

Animate the scrollbar. By default, the value is set to `true`.

## select

Brings the target element to its selected state. The state is automatically reflected in everything that is dependent on the element which changed state (for example, behaviors and observing elements).

#### Attributes

##### **b:target**

Specify an XPath expression pointing to the target element. The default value is `.` (self).

### Implementation

In the following example, the behavior contains three event handlers: when an element that uses the behavior is clicked, a `select` action is triggered that selects the current element, and a `deselect` action that deselects its sibling elements. The actions trigger the

`select` event handler that shows the child `div` element of the element that is selected, and the `deselect` event handler that hides the child `div` element of all deselected elements.

```
<s:behavior b:name="open1child">
    <s:event b:on="command">
        <s:task b:action="select" b:target=". " />
        <s:task b:action="deselect" b:target="following-sibling::div |
preceding-sibling::div" />
    </s:event>
    <s:event b:on="select">
        <s:task b:action="show" b:target="div" />
    </s:event>
    <s:event b:on="deselect">
        <s:task b:action="hide" b:target="div" />
    </s:event>
</s:behavior>
<div>
    <div b:behavior="open1child" class="myClass">Seven Samurai&gt;&gt;
        <div style="font-size: 10px; display: none">A film by....</div>
    </div>
    <div b:behavior="open1child" class="myClass">Life is Beautiful&gt;&gt;
        <div style="font-size: 10px; display: none">A film by...</div>
    </div>
    <div b:behavior="open1child" class="myClass">Citizen Kane&gt;&gt;
        <div style="font-size: 10px; display: none">A film by</div>
    </div>
</div>
```

## select-deselect

Toggles the target element's state between the selected and Deselected states. The state is automatically reflected in everything that is dependent on the element which changed state (for example, behaviors and observing elements).

### Attributes

#### b:target

Specify an XPath expression pointing to the target element. The default value is `.` (self).

### Implementation

In the following example, the behavior contains three event handlers: when an element that uses the behaviors is clicked, a `select-deselect` action is triggered that toggles the element's state from selected to deselected, and triggers respectively either the `select` event handler that shows the child `div` element, or the `deselect` event handler that hides the child `div` element.

```
<style type="text/css"> .myClass { font-size: 16px; line-height: 22px; border: 1px solid
#CCC; width: 200px }</style>
<s:behavior b:name="openChildren">
    <s:event b:on="command">
        <s:task b:action="select-deselect" b:target=". " />
    </s:event>
    <s:event b:on="select">
        <s:task b:action="show" b:target="div" />
    </s:event>
    <s:event b:on="deselect">
        <s:task b:action="hide" b:target="div" />
    </s:event>
</s:behavior>
<div b:behavior="openChildren" class="myClass">Seven Samurai&gt;&gt;
    <div style="font-size: 10px; display: none">A film by....</div>
</div>
<div b:behavior="openChildren" class="myClass">Life is Beautiful&gt;&gt;
    <div style="font-size: 10px; display: none">A film by...</div>
</div>
<div b:behavior="openChildren" class="myClass">Citizen Kane&gt;&gt;
```

```
<div style="font-size: 10px; display: none">A film by</div>
</div>
```

## send

Sends a form, or XML (a BXML element, or set of elements) to the server.

If the `send` command is used to target an element using the `b:source` attribute, and the `b:source` is not pointing to a `form` element, the element and all descending elements are converted to XML and sent using the `HTTP POST` command with the `text/xml` encoding. An XML-enabled server can then load the XML directly into an XML document (the location of the server file is defined using the `b:url` attribute).

If you define a `submit` event handler that is triggered when a form is submitted, you must use the `send` command to actually submit the form. The form is then submitted as a standard HTML submit using the `action` and `method` attributes defined on the form that indicate respectively the name of the file to send the form content to, and the submit method (`post` or `get`) to use.

You can use this command on elements in the *BXML Space* and *HTML Space*.

This command does not trigger a corresponding event.

The `send` command does not work across domains. To send a form or XML to an application that does not reside on the same server, you have to write your own server-side proxy.

### Attributes

#### `b:source`

Specify the source element to use for the action using an XPath expression. The default is the current element.

Note you can also specify a variable for `b:source`, for example `b:source="$myvar"`.

#### `b:destination`

Specify an XPath expression pointing to the receiving/related element. The `b:destination` of a `load`, `xsl-transform`, `send`, or `submit` command can be a variable, for example `b:destination="$myvar"`, in which case the variable is resolved to a String, or it can target an element in the BXML tree, in which case it is resolved to BXML.

#### `b:url`

Specify the path of the file to load, relative to the startup file of your Backbase application.

#### `b:autoroot`

By default, the generated XML is wrapped in a `b:backbase` tag to create valid XML. Set `b:autoroot="false"` to not wrap the XML in a backbase tag.

## set

Sets the attribute or text node targeted with the `b:target` attribute.

You can use this command on elements in the *BXML Space* and *HTML Space*.

### Attributes

#### [Required] `b:target`

Specify an XPath expression pointing to the target element. The default value is `.` (self).

#### [Required] `b:value`

A string containing a stated value.

## Implementation

In the following example, `mouseenter` and `mouseleave` event handlers are defined within

a behavior used by the `b:box` controls. When the mouse enters the box, the `set` command sets style attributes for font-size, box dimensions, and background color. When the mouse leaves the box, the style dimensions are set back to their original values (note that the `b:style` attribute is for styling the inner content of the `b:box` control).

```
<s:behavior b:name="setcommand">
  <s:event b:on="mouseenter">
    <s:task b:action="set" b:target="//b:box/@style" b:value="width: 250px; height:
100px;" />
      <s:task b:action="set" b:target="//b:box/@b:style" b:value="font-size: large;
background-color: #CCFF00" />
    </s:event>
    <s:event b:on="mouseleave">
      <s:task b:action="set" b:target="//b:box/@style" b:value="width: 250px; height:
30px;" />
        <s:task b:action="set" b:target="//b:box/@b:style" b:value="font-size: xx-small;
background-color: #CCFF00" />
    </s:event>
  </s:behavior>
<b:box b:behavior="setcommand" style="width: 250px; height: 30px" b:style="font-size:
xx-small; background-color: #CCFF00">      "Scarface developed a cult following among younger
audiences."</b:box>
<b:box b:behavior="setcommand" style="width: 250px; height: 30px" b:style="font-size:
xx-small; background-color: #CCFF00">      "Scarface developed a cult following among younger
audiences."</b:box>
```

## setcookie

Sets a cookie.

### Attributes

**[Required] `b:name`**

Specify a logical unique name with which to identify an element.

**[Required] `b:value`**

A string containing a stated value.

**`b:days`**

Specify the number of days before the cookie expires.

### Implementation

In the following example, when you resize the element the `resize` event handler uses the `setcookie` command to set cookies for the element's style height and width properties (`style::` is a Backbase XPath axis for targetting individual style property values). A `construct` event handler is also defined that first tests whether the cookies exist (using the Backbase XPath function `cookie()`), and if so it sets the width and height properties to the values stored in the cookie:

```
<div b:resize="all" style="position: absolute; left: 10px; top: 10px; border: solid 2px red;
width: 200px; height: 200px;">
  <p>This element is resizable.</p>
  <p>When you resize the element, it sets a cookie.</p>
  <p>Refresh the page to see element height and width values stored in the cookie being
used to reconstruct the element to its last used dimensions.</p>
  <s:event b:on="resize">
    <s:task b:action="setcookie" b:name="div-height" b:value="{style::height}" b:days="3"
/>
    <s:task b:action="setcookie" b:name="div-width" b:value="{style::width}" b:days="3"
/>
  </s:event>
  <s:event b:on="construct">
    <s:task b:test="cookie('div-height')" b:action="set" b:value="{cookie('div-height')}"
b:target="style::height" />
    <s:task b:test="cookie('div-width')" b:action="set" b:value="{cookie('div-width')}"
b:target="style::width" />
  </s:event>
</div>
```

## setpanelset

Sets new values to an existing `b:panelset` tag. You can specify new panelset layout values for `b:rows` or `b:cols` attributes. If you add columns of rows, you must also add a `b:panel` using for example a `s:render` in which you define the panel to be added. Similarly, if you specify a new value with fewer columns or rows, you must also remove a panel from the panelset, for example by using the `remove` command.

### Attributes

#### `b:target`

Specify an XPath expression pointing to the target element. The default value is `.` (self).

#### `b:rows`

Specify row dimensions defining values or a space separated set of values in `%`, `px`, `pc`, `pt`, `em`, `ex`, `in`, `cm` or `mm`. You can also use the wildcard asterisk (\*) sign to fill the remaining space, in which case all values before or after the wildcard must use the same unit of measurement. For example, `"50px 100px *"`

#### `b:cols`

Specify column dimensions defining values or a space separated set of values in `%`, `px`, `pc`, `pt`, `em`, `ex`, `in`, `cm` or `mm`. You can also use the wildcard asterisk (\*) sign to fill the remaining space, in which case all values before or after the wildcard must use the same unit of measurement. For example, `"50px 100px *"`

## Implementation

In the following example, clicking the `<a id="addPanel">` link; renders a panel in the panelset whose id is `addRow`, adds an additional row to the said panelset, and then removes the link element.

```

<style type="text/css">      .content_wrapper {      width: 100%;      height: 100%;  
border-width: 1px;      border-style: solid; </style>  
<b:panelset b:cols="25% *">  
    <b:panel id="panel_left" b:resize="e" b:minwidth="30px" b:resizeconstraint="..">  
        <div class="content_wrapper">  
            <p>Panel 1</p>  
        </div>  
    </b:panel>  
    <b:panelset id="addRow" b:rows="40% *">  
        <b:panel id="panel_top" b:resize="s" b:minheight="30px" b:resizeconstraint="..">  
            <div class="content_wrapper">  
                <p>Panel 2</p>  
                <a id="addPanel">Add a custom panel...  
                    <s:event b:on="command">  
                        <s:task b:action="render" b:render="render"  
b:destination="id('addRow')" b:mode="aslastchild" />  
                        <s:task b:action="setpanelset" b:target="id('me')" b:rows="33% 33%  
34%" />  
                            <s:task b:action="remove" b:target=".." />  
                        </s:event>  
                        <s:render b:name="render">  
                            <b:panel id="customPanel">  
                                <div style="content_wrapper">  
                                    <p>Panel 4: Custom panel contents....</p>  
                                </div>  
                            </b:panel>  
                        </s:render>  
                    </a>  
                </div>  
            </b:panel>  
            <b:panel id="panel_bottom">  
                <div class="content_wrapper">  
                    <p>Panel 3</p>  
                </div>  
            </b:panel>  
        </b:panelset>
    
```

```
</b:panelset>
```

## setstatus

Displays the string value specified by the `b:value` attribute in the browser's status bar.

### Attributes

**[Required] b:value**

A string containing a stated value.

### Implementation

```
<div b:action="setstatus" b:value="You have a working Backbase installation"> Click here  
to send a message to the browser status bar... </div>
```

## setstruct

Sets the target element's HTML structure to the name specified in the `b:value` attribute.

### Attributes

**[Required] b:value**

A string containing a stated value.

**b:target**

Specify an XPath expression pointing to the target element. The default value is `.` (self).

## settext

Sets the text node of the target element to the specified value. If no text node exists on the target element(s), a text node is created.

If the target contains more than one text node, only the first text node is replaced. If you want to target several text nodes, use the `set` command

### Attributes

**b:target**

Specify an XPath expression pointing to the target element. The default value is `.` (self).

**[Required] b:value**

A string containing a stated value.

### Implementation

```
<b:box>Click me to replace the text...  
  <b>sdsd</b>  
sdsdsd  
  <s:event b:on="click">  
    <s:task b:action="settext" b:value="Welcome to Backbase!" />  
  </s:event>  
</b:box>
```

## setttitle

Changes the browser window title to the string specified in the `b:value` attribute.

### Attributes

**[Required] b:value**

A string containing a stated value.

### Implementation

```
<div b:action="setttitle" b:value="Backbase RIA applications..."> Click here to change  
the browser window title... </div>
```

## show

Shows the target element by setting its `display` style property.

The commands `show` and `show-hide` are designed to work on block level elements: the commands overrule the CSS style `display:none` defined in a class for the elements that you want to show. To use on non-block-level elements, you must NOT specify the `display:none` style in your stylesheet (using inline styles IS permitted).

You can use this command on elements in the *BXML Space* and *HTML Space*.

### Attributes

#### b:target

Specify an XPath expression pointing to the target element. The default value is `.` (self).

### Implementation

In the following example, clicking the link shows the `p` element moves down.

```
<a b:action="show" b:target=".../p[span]">Click here to show the text below.</a>
<p style="display: none">
    <span>This text will be shown</span>
</p>
<p>This moves down.</p>
```

## show-hide

Shows or hides the target element by toggling its `display` style property depending on its previous state.

The commands `show` and `show-hide` are designed to work on block level elements: the commands overrule the CSS style `display:none` defined in a class for the elements that you want to show. To use on non-block-level elements, you must NOT specify the `display:none` style in your stylesheet (using inline styles IS permitted).

You can use this command on elements in the *BXML Space* and *HTML Space*.

### Attributes

#### b:target

Specify an XPath expression pointing to the target element. The default value is `.` (self).

### Implementation

In the following example, clicking the link shows/hides works on the element with the ID `sbox-showhide` and all `a` elements (the `show` less link is not initially hidden (`display:none`), so only one of the link elements is visible at any one time):

```
<b:box>
    <h3>The Seven Samurai (1954)</h3>
    <p>A movie by Akira Kurosawa starring Takashi Shimura and Toshiro Mifune.</p>
    <p style="display: none" id="sbox-showhide">The film takes place in war-ridden 16th century Japan, where a village of farmers hire seven ronin (lordless samurai) to fight for them against a band of marauding robbers. The Seven Samurai is widely regarded as one of the greatest Japanese films ever made.</p>
    <p b:action="show-hide" b:target="id('sbox-showhide') | a">
        <a style="display:none">&lt;&lt;&lt; Show less</a>
        <a>Read more &gt;&gt;&gt;</a>
    </p>
</b:box>
```

## sort

Sorts the targeted elements. By default, sorting is in alphabetical order. The attributes you need to use to implement sorting depend on the element you want to sort:

- Tables - to sort the rows in the `tbody` section of a `table`, its target can only be a `td` or a `th` in the table's `thead`. The table rows are sorted based on the contents of the cells or (if present) on the numerical value in the `b:sortvalue` property of the cell.
- Other elements are sorted based on one of its attributes specifies by the `b:attribute`.

#### Attributes

##### `b:target`

Specify an XPath expression pointing to the target element. The default value is `.` (self).

##### `b:attribute`

Specify the name of the attribute that is used as the sort value.

##### `b:reverse`

Use with `b:sortvalue` attribute to determine whether the values are sorted by default in reverse order. (You can toggle between forward and reverse sorting.)

##### `b:sortvalue`

Specify a string value that determines the cell is sorted by this value instead of by its contents.

##### `b:type`

By default, when a `sort` action is done, the BPC evaluates whether sorting should be performed using an integer or a string value. Setting `b:type="string"` forces the `sort` action to sort according to the string value.

##### `b:source`

Specify the source element to use for the action using an XPath expression. The default is the current element.

Note you can also specify a variable for `b:source`, for example `b:source="$myvar"`.

## Implementation

In the following example, clicking the table headers triggers the `sort` command to sort the table cell contents in that column.

Note that the `sort` command sorts prefixed numerical values by the numerical value itself by using the value in the `b:sortvalue` attribute of the table cell elements.

```
<table style="width:300px; border: 1px solid #D0E9FD;" cellspacing="0">
  <caption>Click a column head to sort</caption>
  <thead style="background-color: #D0E9FD;">
    <tr>
      <th b:action="sort" b:cursor="pointer" b:reverse="false">Item</th>
      <th b:action="sort" b:cursor="pointer" b:reverse="true">Price</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Pen</td>
      <td b:sortvalue="1.75">EUR 1,75</td>
    </tr>
    <tr>
      <td>CD-Recordables (50pcs)</td>
      <td b:sortvalue="20.00">EUR 20,-</td>
    </tr>
    <tr>
      <td>Stamps (25pcs)</td>
      <td b:sortvalue="15.00">EUR 15,-</td>
    </tr>
  </tbody>
</table>
```

In the following example, clicking the link triggers the `sort` action to sort the list items according to the `value` attribute:

```
<a b:action="sort" b:source="//ul/li" b:attribute="value" b:target="//ul">Sort List in
Alphabetical Order</a>
<ul>
    <li value="Ba">Bananas</li>
    <li value="Da">Damsens</li>
    <li value="Ca">Carrots</li>
    <li value="Ap">Apples</li>
</ul>
```

## store

Stores the position and dimensions of the target element, which can be restored using the `restore` event. The `store` and `restore` commands are used, for example, for the `b:window` control to implement the window minimize and maximize buttons.

You can use this command on elements in the *BXML Space* and *HTML Space*.

### Attributes

#### `b:target`

Specify an XPath expression pointing to the target element. The default value is `.` (self).

## Implementation

In the following example, clicking the first link stores the box dimensions, and the second link restores box to its last stored dimensions (after the `box` has been resized).

```
<a b:action="store" b:target="//b:box">Click to store box dimensions</a>
<span>|</span>
<a b:action="restore" b:target="//b:box">Click to restore box dimensions</a>
<b:box b:resize="all" b:style="background-color: #CCFF00">          "You can resize this
box....." </b:box>
```

## string2xml

Converts a String value to XML. If you specify only a `b:variable`, it uses it as the source and the target; it converts the String value contained in the variable to XML and returns the result back into the variable. The `string2xml` command is useful, for example, when you load in data using the `load` command and want to convert the resulting String (the result of a `load` action is always a String), to XML for templating.

### Attributes

#### [Required] `b:variable`

Specify the unique name of a variable, that has been declared using the `s:variable` tag, in which the result of the transformation is stored. The variable name must be prefixed by a \$ sign, for example `b:variable="$data"` where `data` is the variable name.

#### `b:source`

Specify the source element to use for the action using an XPath expression. The default is the current element.

Note you can also specify a variable for `b:source`, for example `b:source="$myvar"`.

#### `b:namespaces`

Define a space-separated list of namespaces that are used to insert namespace declarations into the document. For example `b:namespace="s b"` results in:

```
xmlns:s="http://www.backbase.com/s"                                         xm-
lns:b="http://www.backbase.com/b"
```

## Implementation

In the following example, the `load` commands load data from an XML file, and an XSLT, and stores the result in a variables. As the result of the `load` command is a String, you

need to use the `string2xml` command to convert the variables back into XML where they can then be used for transformation (see also Browser Templating in the Manual PDF):

```
<div b:behavior="browsertemplating"></div>
<s:behavior b:name="browsertemplating" b:focusgroup="true">
    <s:event b:on="construct">
        <s:variable b:name="data" b:scope="global" />
        <s:variable b:name="xslt" b:scope="global" />
        <s:task b:action="load" b:destination="$xslt" b:url="table.xslt" />
        <s:task b:action="load" b:destination="$data" b:url="films.xml" />
        <s:task b:action="string2xml" b:variable="$xslt" />
        <s:task b:action="string2xml" b:variable="$data" />
        <s:task b:action="xsl-transform" b:stylesheet="$xslt" b:datasource="$data"
b:destination"." />
    </s:event>
</s:behavior>
```

## Submit

Used to submit a form. If you define an event handler for the `submit` event that is triggered when the form is submitted, for example to validate the form, you must also issue a `send` command to actually submit the form.

### Attributes

#### `b:target`

Specify an XPath expression pointing to the target element. The default value is `.` (self).

## Implementation

In the following example, a `submit` command is defined within a button control that targets the form element, using its ID, to be submitted:

```
<form id="main" b:destination="." b:mode="replace" action="response.php" method="post">
    <table>
        <tr>
            <td>Name</td>
            <td>
                <input name="name" type="text" size="40" />
            </td>
        </tr>
        <tr>
            <td>Data</td>
            <td>
                <textarea name="data" />
            </td>
        </tr>
    </table>
    <b:button b:action="submit" b:target="id('main')">Send</b:button>
</form>
```

## tile

Places targeted elements in a tiled formation. The targeted elements need to be positioned absolutely. The width and height available for the tiling is determined by the element's offset parent. The offset parent is the first parent element that has `position:relative` or `position:absolute` defined as CSS property.

When you use the `tile` command, ensure that ancestor elements do not have the `style="display:none"` property set. The command can experience problems with drawing its target element if this property is set.

### Attributes

#### [Required] `b:target`

Specify an XPath expression pointing to the target element. The default value is `.` (self).

**b:orientation**

Specify the orientation of elements displayed in a tile formation. The default is **cols**.

**Values:**

**cols**  
**rows**

**b:max**

Specify a maximum size in pixels.

**Implementation**

The following example uses the **tile** command to tile a series of blocks displaying the text BACKBASE; click on the links to arrange these blocks vertically or horizontally:

```
<style type="text/css">
    .logoletter {
        width: 20px;
        height: 10px;
        position: absolute;
        font-size: 22px;
        line-height: 22px;
        font-weight: bold;
        color: #C00;
        border: 1px solid #CCC;
    } </style>

<s:behavior b:name="tile">
    <s:event b:on="select">
        <s:task b:action="fxtile" b:target="div" b:orientation="cols" />
    </s:event>
    <s:event b:on="deselect">
        <s:task b:action="fxtile" b:target="div" b:orientation="rows" />
    </s:event>
</s:behavior>
<p>
    <a b:action="select" b:target="id('container')">Tile "Backbase" by column</a>
    <span>|</span>
    <a b:action="deselect" b:target="id('container')">Tile "Backbase" by row</a>
</p>
<div style="position: relative; height: 200px;" b:behavior="tile" id="container">
    <div id="div1" class="logoletter">B</div>
    <div id="div2" class="logoletter">A</div>
    <div id="div3" class="logoletter">C</div>
    <div id="div4" class="logoletter">K</div>
    <div id="div5" class="logoletter">B</div>
    <div id="div6" class="logoletter">A</div>
    <div id="div7" class="logoletter">S</div>
    <div id="div8" class="logoletter">E</div>
</div>
```

**transform**

Transforms and renders XML data (the datasource) using a stylesheet. This command does not trigger a corresponding event.

**Attributes****[Required] b:stylesheet**

Invoke a template stylesheet that defines the starting point of a block of processing/transformation rules.

If you use Backbase templating (**transform** command), the **b:name** attribute of the **s:stylesheet** tag is stored in a variable that you can access using the syntax **\$name**.

If you use Browser templating (**xsl-transform** command), the **b:stylesheet** attribute must refer to variable that contains XML.

**[Required] b:datasource**

Specify the XML datasource.

When the XML is part of the main BXML body, wrapped in an `s:xml` tag, the `b:name` of the datasource is stored in a variable that you can access using the syntax `$name`.

If you use Browser templating (`xsl-transform` command), the `b:datasource` attribute must refer to variable that contains XML.

#### `b:destination`

Specify an XPath expression pointing to the receiving/related element. The `b:destination` of a `load`, `xsl-transform`, `send`, or `submit` command can be a variable, for example `b:destination="$myvar"`, in which case the variable is resolved to a String, or it can target an element in the BXML tree, in which case it is resolved to BXML.

#### `b:mode`

Define where a node is inserted relative to the receiving element. By default, the value is set to `replacechildren`.

Values:

- after
- asfirstchild
- aslastchild
- before
- replace
- replacechildren

## Implementation

In the following example, when you click the `div` the BPC transforms the inline XML (catalog datasource) using the stylesheet template (`products`) and renders the output in the `div`. Note that the `b:name` of the stylesheet and datasource are available as variables and accessible by prefixing the name value with a dollar.

```
<s:xml b:name="catalog">
  <products>
    <product name="Apples">Apples</product>
    <product name="Oranges">Oranges</product>
    <product name="Pears">Pears</product>
    <product name="Bananas">Bananas</product>
  </products>
</s:xml>
<s:stylesheet b:name="products">
  <s:template b:match="/">
    <div style="border: 1px solid red; width:100px;">
      <s:apply-templates b:select="*/product" />
    </div>
  </s:template>
  <s:template b:match="product">
    <div>
      <s:attribute b:name="id" b:select="@name" />
      <s:value-of b:select="@name" />
    </div>
  </s:template>
</s:stylesheet>
<div id="output">Click here to view products...
  <s:event b:on="command">
    <s:task b:action="transform" b:stylesheet="$products" b:datasource="$catalog"
b:destination="id('output')" b:mode="aslastchild" />
  </s:event>
</div>
```

## trigger

Triggers a user-defined custom event, or an existing BPC event. The event handler for the triggered event is defined in the element itself, or in a behavior. You define a custom event simply by giving it a name, for example `b:on="myEvent"` and defining an event handler that defines the tasks that are carried out when the event is triggered. You trigger the custom event using the syntax `b:action="trigger" b:event="myEvent"`

**b:target=". . ."**, where the target specifies the element in which the event is triggered. This command does not trigger a corresponding event.

#### Attributes

##### [Required] **b:event**

Indicate the event that is triggered. The event can be a BPC event, such as `change` or `select`, or a user-defined custom event, such as `changeView`.

##### **b:target**

Specify an XPath expression pointing to the target element. The default value is `. (self)`.

##### **b:usebehavior**

Trigger the event defined in a given `s:behavior` tag, effectively calling the behavior's event handler.

##### **b:delay**

Delay the triggering of the event by specify a duration in milliseconds.

#### Implementation

In the following example, the `s:behavior b:name="write"` defines the custom events `press_a_key`, `press_c_key`, and `press_e_key`. The `s:whenactive` tag in the behavior defines the keys that are active within the `div` element. The `s:keys` defined within the `s:whenactive` trigger the custom events using the `trigger` command.

```
<s:behavior b:name="write">
    <s:event b:on="press_a_key">
        <s:task b:action="alert" b:value="'A' is a hot key!" />
    </s:event>
    <s:event b:on="press_c_key">
        <s:task b:action="set" b:target="//textarea/@style" b:value="background-color: green; color: white; width: 300px; height: 100px" />
    </s:event>
    <s:event b:on="press_e_key">
        <s:task b:action="visible-invisible" b:target="//textarea" />
    </s:event>
    <s:whenactive>
        <s:keys b:keys="a" b:action="trigger" b:event="press_a_key" b:target="id('dest')"/>
        <s:keys b:keys="c" b:action="trigger" b:event="press_c_key" b:target="id('dest')"/>
        <s:keys b:keys="e" b:action="trigger" b:event="press_e_key" b:target="id('dest')"/>
    </s:whenactive>
</s:behavior>


The following is an example of the b:delay attribute to postpone the triggering of the event:



```
<s:behavior b:name="me">
    <s:event b:on="command">
        <s:task b:action="trigger" b:event="xx" b:delay="5000" />
    </s:event>
    <s:event b:on="xx">
        <s:task b:action="alert" b:value="Should appear after 5 seconds" />
    </s:event>
</s:behavior>
<b:box>
    <div b:behavior="me">Click me and wait...</div>
</b:box>
```



## visible



Sets the visibility style property of the target element to visible.



You can use this command on elements in the BXML Space and HTML Space.



#### Attributes



©2004-2006 Backbase B.V., All Rights Reserved.



139


```

**b:target**

Specify an XPath expression pointing to the target element. The default value is . (self).

Implementation

```
<a b:action="visible" b:target="../b:box[1]">Click here to make the element below visible</a>
<b:box style="visibility:hidden; width: 150px">When you click the link, this box appears...</b:box>
<b:box style="width: 150px">The position of this element remains the same regardless....</b:box>
```

**visible-invisible**

Toggles the `visibility` style property of the target element to `visible` or `hidden`, rendering it visible or invisible according to the current value of the style property.

You can use this command on elements in the *BXML Space* and *HTML Space*.

Attributes**b:target**

Specify an XPath expression pointing to the target element. The default value is . (self).

Implementation

```
<a b:action="visible-invisible" b:target="../b:box">Click here to toggle the box elements visibility property</a>
<b:box style="width: 150px">This box is by default visible</b:box>
<b:box style="visibility:hidden; width: 150px">This box is by default hidden</b:box>
```

**xml2string**

Converts an XML node-set to a String. If you specify only a `b:variable`, it uses it as the source and the target; it converts the XML contained in the variable to a String and returns the result back into the variable.

Attributes**[Required] b:variable**

Specify the unique name of a variable, that has been declared using the `s:variable` tag, in which the result of the transformation is stored. The variable name must be prefixed by a \$ sign, for example `b:variable="$data"` where `data` is the variable name.

**b:source**

Specify the source element to use for the action using an XPath expression. The default is the current element.

Note you can also specify a variable for `b:source`, for example `b:source="$myvar"`.

Implementation

```
<b:box>
  <div>Click here to see transformations
    <s:event b:on="command">
      <s:variable b:name="data" />
      <s:task b:action="load" b:url="data/films.xml" b:destination="$data" />
      <s:task b:action="string2xml" b:variable="$data" />
      <s:task b:action="remove" b:target="$data//film except $data/films/film[1]" />
      <s:task b:action="xml2string" b:variable="$data" />
      <s:task b:action="move" b:source="$data" b:destination=".." b:mode="replace" />
    </s:event>
  </div>
</b:box>
```

Where `films.xml` contains the following:

```
<?xml version="1.0" ?>
```

```

<films>
  <film id="1">
    <title>The Fifth Element</title>
    <release>1997</release>
    <description>250 years in the future, life is threatened by the arrival of Evil, and
only "The Fifth Element" can save the world.</description>
  </film>
  <film id="2">
    <title>Blade Runner</title>
    <release>1982</release>
    <description>In a future where genetically manufactured beings called replicants are
used for dangerous and degrading work in Earth's off-world colonies, specialist police units
-           Blade Runners - hunt down and retire (kill) escaped replicants on
Earth.</description>
  </film>
  <film id="3">
    <title>American Beauty</title>
    <release>1999</release>
    <description>A 1999 drama film that explores themes of love, freedom, self
liberation, family, and the American Dream.</description>
  </film>
</films>

```

## xsl-transform

Instructs the client browser to transform and render XML data (the datasource) using a stylesheet.

This command does not trigger a corresponding event.

### Attributes

#### [Required] b:stylesheet

Invoke a template stylesheet that defines the starting point of a block of processing/transformation rules.

If you use Backbase templating (`transform` command), the `b:name` attribute of the `s:stylesheet` tag is stored in a variable that you can access using the syntax `$name`.

If you use Browser templating (`xsl-transform` command), the `b:stylesheet` attribute must refer to variable that contains XML.

#### [Required] b:datasource

Specify the XML datasource.

When the XML is part of the main BXML body, wrapped in an `s:xml` tag, the `b:name` of the datasource is stored in a variable that you can access using the syntax `$name`.

If you use Browser templating (`xsl-transform` command), the `b:datasource` attribute must refer to variable that contains XML.

#### [Required] b:destination

Specify an XPath expression pointing to the receiving/related element. The `b:destination` of a `load`, `xsl-transform`, `send`, or `submit` command can be a variable, for example `b:destination="$myvar"`, in which case the variable is resolved to a String, or it can target an element in the BXML tree, in which case it is resolved to BXML.

#### b:mode

Define where a node is inserted relative to the receiving element. By default, the value is set to `replacechildren`.

##### Values:

- after
- asfirstchild
- aslastchild
- before
- replace
- replacechildren

## Implementation

In the following example, when the application is constructed the datasource `films.xml`

and stylesheet `table.xslt` are loaded and put into globally-declared variables, then converted from Strings to XML documents that allows the browser to perform the transformation:

```
<div b:behavior="browsertemplating"></div>
<s:behavior b:name="browsertemplating" b:focusgroup="true">
  <s:event b:on="construct">
    <s:variable b:name="data" b:scope="global" />
    <s:variable b:name="xslt" b:scope="global" />
    <s:task b:action="load" b:destination="$data" b:url="films.xml" />
    <s:task b:action="load" b:destination="$xslt" b:url="table.xslt" />
    <s:task b:action="string2xml" b:variable="$xslt" />
    <s:task b:action="string2xml" b:variable="$data" />
    <s:task b:action="xsl-transform" b:stylesheet="$xslt" b:datasource="$data"
b:destination"." />
  </s:event>
</s:behavior>
```

## **zsort**

Moves the targeted element to the last position of its parent's children. When there are overlapping elements in HTML, and no z-order is defined, the positioning in the source code determines which element is visible. This command does not trigger a corresponding event.

### Attributes

#### **b:target**

Specify an XPath expression pointing to the target element. The default value is `.` (self).

## Implementation

```
<div>Click a list item to move to bottom of list:</div>
<ul>
  <li b:action="zsort">Apples</li>
  <li b:action="zsort">Bananas</li>
  <li b:action="zsort">Cherries</li>
  <li b:action="zsort">Damson</li>
</ul>
```

## BXML Events

### active

The `active` event takes place on an element when it, or a child node, gets focus.

#### Implementation

In the following example, when you click within the `div` elements; the active elements are displayed in green and inactive elements in red:

```
<s:behavior b:name="showactiveelements">
    <s:event b:on="construct">
        <s:setstyle b:padding="10px" b:border="solid 1px black" />
    </s:event>
    <s:event b:on="active">
        <s:setstyle b:background-color="green" />
    </s:event>
    <s:event b:on="inactive">
        <s:setstyle b:background-color="red" />
    </s:event>
</s:behavior>
<div b:behavior="showactiveelements" b:focusroot="true" b:focusitem="true">FocusRoot
    <div b:behavior="showactiveelements" b:focusgroup="true" b:focusitem="true">Focusgroup 2
        <div b:behavior="showactiveelements" b:focusitem="true">Focusitem 2.1</div>
        <div b:behavior="showactiveelements" b:focusitem="true">Focusitem 2.2</div>
        <div b:behavior="showactiveelements" b:focusitem="true">Focusitem 2.3</div>
        <div b:behavior="showactiveelements" b:focusitem="true">Focusitem 2.4</div>
    </div>
</div>
```

### blocked-mousedown

The `blocked-mousedown` event occurs instead of a normal `mousedown` event when the left mouse button is pressed on an element that is blocked from receiving input events. An element is blocked from receiving events when it or its parent has the `b:eventblock` attribute set to `true`.

### blur

The `blur` event occurs when a form element, or `b:focusitem`, loses focus.

#### Implementation

In the following example, use the Tab key to navigate between focusgroups, and the arrow keys to navigate individual focusitems in the groups. When an element gets focus, the `focus` event is triggered and the color of its text is changed:

```
<s:behavior b:name="focusobject">
    <s:event b:on="blur">
        <s:setstyle b:color="#51698E" b:opacity="80" />
    </s:event>
    <s:event b:on="focus">
        <s:setstyle b:opacity="100" b:color="red" />
    </s:event>
</s:behavior>
<div b:behavior="focusobject" b:focusgroup="true">Films
    <ul>
        <li>
            <a b:behavior="focusobject" b:focusgroup="false" b:focusitem="true">Seven
Sumarai</a>
        </li>
        <li>
            <a b:behavior="focusobject" b:focusgroup="false" b:focusitem="true">The Huducker
Proxy</a>
        </li>
        <li>
            <a b:behavior="focusobject" b:focusgroup="false" b:focusitem="true">Life is
```

```

Beautiful</a>
    </li>
</ul>
</div>
<div b:behavior="focusobject" b:focusgroup="true">Authors
<ul>
    <li>
        <a b:behavior="focusobject" b:focusgroup="false" b:focusitem="true">Paul
Auster</a>
        </li>
        <li>
            <a b:behavior="focusobject" b:focusgroup="false" b:focusitem="true">Henry
Miller</a>
            </li>
            <li>
                <a b:behavior="focusobject" b:focusgroup="false" b:focusitem="true">Joseph
Conrad</a>
                </li>
            </ul>
</div>

```

## change

The `change` event occurs when a user leaves the current input field that has focus and whose value has been modified since gaining focus. It is used to validate fields.

The event applies to input elements such as `input`, `select`, `textarea`, `b:slider`, `b:spinner`.

### Implementation

In the following example, if you enter a value less than 18 in the input field, when you leave the field the `change` event is triggered and a basic validation is performed:

```

<s:behavior b:name="x">
    <s:event b:on="change">
        <s:choose>
            <s:when b:test="@value lt 18">
                <s:task b:action="alert" b:value="You must be older than 18 to participate"
/>
            </s:when>
            <s:otherwise />
        </s:choose>
    </s:event>
</s:behavior>
<div>Enter your age:
    <input b:behavior="x" type="input" value="" />
</div>

```

## child-lost

The `child-lost` event occurs on a parent element of a child node (or nodes) that has been dragged-and-dropped, or moved, to another location in the BXML tree.

The following variables are available for this action:

- `_source` - contains the parent element of the element on which the action is being performed
- `_element` - contains the element on which the action is being performed
- `_target` - contains the element which is the target of the action
- `_sourceIndex` - contains the child element index of the element on which the action is being performed relative to its parent; that is, whether it was the first, second, third, fourth, and so on child of the parent

### Implementation

In the following example, you can click an item in the list to move it from one list to an-

other. When you do so, a `child-lost` event is triggered; an event handler is defined for `child-lost` that activates an alert box to inform you of the element that was moved, from which element and to where (using the available `child-lost` variables):

```
<div id="container">
    <p>Click the articles to move them to your shopping list...</p>
    <b:box b:style="width: 400px; height: 100px">
        <p>Product List:</p>
        <ul id="products list">
            <s:event b:on="child-lost">
                <s:task b:action="alert" b:value="{concat ('The element ', $_element, ' was
moved from the ', $_source/@id, ' to the ', $_target/@id)}" />
            </s:event>
            <li b:action="move" b:source=". " b:destination="id('container')//ul except ...
b:mode="aslastchild">Cheese</li>
            <li b:action="move" b:source=". " b:destination="id('container')//ul except ...
b:mode="aslastchild">Eggs</li>
            <li b:action="move" b:source=". " b:destination="id('container')//ul except ...
b:mode="aslastchild">Milk</li>
        </ul>
    </b:box>
    <br />
    <b:box b:style="width: 400px; height: 150px">
        <p>My Shopping List:</p>
        <ul id="shopping list"></ul>
    </b:box>
</div>
```

## click

The `click` event occurs when the user clicks on an element with the pointing device's left button.

### Implementation

```
<a>Click me!
    <s:event b:on="click">
        <s:task b:action="alert" b:value="Hi!" />
    </s:event>
</a>
```

## close

The `close` event occurs when a node (`b:treelistrow`) in a treelist is closed.

### Implementation

In the following example,

```
<a b:action="select" b:target="id('modal')">Seven Sumurai</a>
<b:modal id="modal">
    <b:modalhead>Confirm Details</b:modalhead>
    <b:modalbody>
        <p>Are your details filled in correctly? If correct, click Yes, otherwise click No
and
        go back and correct as appropriate.</p>
        <b:button b:action="trigger" b:event="close" b:target="id('modal')">Close
Window</b:button>
    </b:modalbody>
</b:modal>
```

## command

The `command` event occurs when the user clicks on an element with the pointing device's left button, or activates it using a key on the keyboard. In most situations, using the `command` event is preferred over using the `click` event.

### Implementation

In the following example, the alert box appears if you click the link element with your left mouse button, or by using a key:

```
<a>Click me!
  <s:event b:on="command">
    <s:task b:action="alert" b:value="Hi!" />
  </s:event>
</a>
```

## construct

The `construct` event occurs when the BPC is building an element in the user interface. It is used, for example, to display a loading message in your startup file for low bandwidth connections. Combining this event with transitions can also ensure your application is loaded smoothly.

### Implementation

In the following example, the `construct` event is used to set the item that gets focus when the page is constructed:

```
<s:event b:on="construct">
  <s:task b:action="focus" b:target="id('age')"/>
</s:event>
<div>
  Name:
  <input type="text" value="" />
  <br />
  <br />
  Age:
  <input id="age" type="text" value="" />
  <br />
  <br />
  Address:
  <input type="text" value="" />
  <br />
  <br />
</div>
```

## copy

The `copy` event occurs when the `copy` action is triggered.

The following variables are available for this action:

- `_source` - contains the parent element of the element on which the action is being performed
- `_element` - contains the element on which the action is being performed
- `_target` - contains the element which is the target of the action
- `_sourceIndex` - contains the child element index of the element on which the action is being performed relative to its parent; that is, whether it was the first, second, third, fourth, and so on child of the parent

### Implementation

In the following example, you can click an item in the list to move it from one list to another:

```
<div id="container">
  <p>Click the articles to move them to your shopping list...</p>
  <b:box b:style="width: 400px; height: 100px">
    <p>Product List:</p>
    <ul id="products list">
      <li b:action="copy" b:source=". " b:destination="id('container')//ul except ..." b:mode="aslastchild">Cheese</li>
      <li b:action="copy" b:source=". " b:destination="id('container')//ul except ..." b:mode="aslastchild">Apples</li>
    </ul>
  </b:box>
</div>
```

```
b:mode="aslastchild">Eggs</li>
    <li b:action="copy" b:source=". " b:destination="id('container')//ul except ..." 
b:mode="aslastchild">Milk</li>
</ul>
</b:box>
<br />
<b:box b:style="width: 400px; height: 150px">
    <p>My Shopping List:</p>
    <ul id="shopping list"></ul>
</b:box>
</div>
```

## dblclick

The `dblclick` event occurs when the pointing device's left button double-clicks an element. Note that a double-click also generates two separate click events.

### Implementation

```
<a>Double-click me!
  <s:event b:on="dblclick">
    <s:task b:action="alert" b:value="Hi!" />
  </s:event>
</a>
```

## deselect

The `deselect` event occurs on an element when it is put in the *deselected* state.

### Implementation

In the following example, the behavior contains three event handlers: when the `div` element is clicked, a `select-deselect` action is triggered that toggles the element's state from selected to deselected, and triggers respectively either the `select` event handler that shows the child `div` element, or the `deselect` event handler that hides the child `div` element.

```
<style type="text/css">.myClass { font-size: 16px; line-height: 22px; border: 1px solid #CCC;
width: 200px }</style>
<s:behavior b:name="openChildren">
  <s:event b:on="command">
    <s:task b:action="select-deselect" b:target=". " />
  </s:event>
  <s:event b:on="select">
    <s:task b:action="show" b:target="div" />
  </s:event>
  <s:event b:on="deselect">
    <s:task b:action="hide" b:target="div" />
  </s:event>
</s:behavior>
<div b:behavior="openChildren" class="myClass">Seven Samurai&gt;&gt;
  <div style="font-size: 10px; display: none">A film by....</div>
</div>
```

## disable

The `disable` event occurs when an element is disabled. When an element is disabled, it cannot respond to user input events. You can enable a disabled element using the `enable` command.

### Implementation

In the following example, clicking the Yes button disables the input fields, clicking the No button enables the fields; when the fields are enabled or disabled, respectively an `enable` or `disable` event is triggered on the elements. Event handlers are defined in a behavior

for these events that trigger an alert box:

```
<s:behavior b:name="enable-disable">
    <s:event b:on="disable">
        <s:task b:action="alert" b:value="The field are now disabled" />
    </s:event>
    <s:event b:on="enable">
        <s:task b:action="alert" b:value="The field are now enabled" />
    </s:event>
</s:behavior>
<div style="width: 300px; height: 130px; padding: 10px">
    <input type="text" value="name" />
    <br />
    <br />
    <input type="text" value="age" />
    <br />
    <br />
    <input type="text" value="address" b:behavior="enable-disable" />
    <br />
    <br />
    Are your details correct?
    <b:button b:action="disable" b:target="//input">Yes</b:button>
    <b:button b:action="enable" b:target="//input">No</b:button>
</div>
```

## drag-deeper

Takes place when an element that is being dragged is moved from a parent element to a child element (down the BXML tree hierarchy). It is triggered on the parent element.

The event is used in conjunction with `b:dragmode="symbol"`. When you specify `b:dragmode="symbol"`, the `bpc_dragSymbol` variable is available in which you can define the appearance of the dragmode.

## drag-drop

Takes place when an element is successfully dropped on a receiving element. It is triggered on the element that is being dropped.

The following drag variables are available:

- `_dragParent` - when an element is dragged, its parent element is stored in the `_dragParent` variable.
- `_dragCurrent` - contains the element that is currently being dragged.
- `_dragTarget` - when a dragged element is dropped, the element on which it is dropped is stored in the `_dragTarget` variable.
- `_sourceIndex` - contains the child element index of the element on which the action is being performed relative to its parent; that is, whether it was the first, second, third, fourth, and so on child of the parent

## drag-enter

Takes place when the element that is being dragged is moved from the parent element to a child element (down the BXML tree hierarchy). It is triggered on the element which the dragged element enters.

The event is used in conjunction with `b:dragmode="symbol"`. When you specify `b:dragmode="symbol"`, the `bpc_dragSymbol` variable is available in which you can define the appearance of the dragmode.

## drag-leave

Takes place when the element that is being dragged is moved from a child element to a parent element. It is triggered on the child element.

The event is used in conjunction with `b:dragmode="symbol"`. When you specify `b:dragmode="symbol"`, the `bpc_dragSymbol` variable is available in which you can define the appearance of the dragmode.

## drag-receive

Takes place on the element which receives the element that has been dragged and dropped.

The following drag variables are available:

- `_dragParent` - when an element is dragged, its parent element is stored in the `_dragParent` variable.
- `_dragCurrent` - contains the element that is currently being dragged.
- `_dragTarget` - when a dragged element is dropped, the element on which it is dropped is stored in the `_dragTarget` variable.
- `_sourceIndex` - contains the child element index of the element on which the action is being performed relative to its parent; that is, whether it was the first, second, third, fourth, and so on child of the parent

## drag-reenter

Takes place when the element that is being dragged is moved from the child element to a parent element (up the BXML tree hierarchy). It is triggered on the element which the dragged element reenters.

The event is used in conjunction with `b:dragmode="symbol"`. When you specify `b:dragmode="symbol"`, the `bpc_dragSymbol` variable is available in which you can define the appearance of the dragmode.

## drag-start

Takes place when the drag action commences. It is triggered on the element that is being dragged.

The following drag variables are available:

- `_dragParent` - when an element is dragged, its parent element is stored in the `_dragParent` variable.
- `_dragCurrent` - contains the element that is currently being dragged.
- `_dragTarget` - when a dragged element is dropped, the element on which it is dropped is stored in the `_dragTarget` variable.

## enable

The `enable` event occurs on an element when the element is enabled.

### Implementation

In the following example, clicking the Yes button disables the input fields, clicking the No button enables the fields; when the fields are enabled or disabled, respectively an `enable` or `disable` event is triggered on the elements. Event handlers are defined in a behavior for these events that trigger an alert box:

```

<s:behavior b:name="enable-disable">
    <s:event b:on="disable" b:async="true">
        <s:task b:action="alert" b:value="The field are now disabled" />
    </s:event>
    <s:event b:on="enable">
        <s:task b:action="alert" b:value="The field are now enabled" />
    </s:event>
</s:behavior>
<div style="width: 300px; height: 130px; padding: 10px">
    <input type="text" value="name" />
    <br />
    <br />
    <input type="text" value="age" />
    <br />
    <br />
    <input type="text" value="address" b:behavior="enable-disable" />
    <br />
    <br />
    Are your details correct?
    <b:button b:action="disable" b:target="//input">Yes</b:button>
    <b:button b:action="enable" b:target="//input">No</b:button>
</div>

```

## focus

The `focus` event occurs when a form element, or `b:focusitem`, gets focus.

### Implementation

In the following example, use the Tab key to navigate between focusgroups, and the arrow keys to navigate individual focusitems in the groups. When an element loses focus, the `blur` event is triggered and the color of its text is changed:

```

<s:behavior b:name="focusobject">
    <s:event b:on="blur">
        <s:setstyle b:color="#51698E" b:opacity="80" />
    </s:event>
    <s:event b:on="focus">
        <s:setstyle b:opacity="100" b:color="red" />
    </s:event>
</s:behavior>
<div b:behavior="focusobject" b:focusgroup="true">Films
    <ul>
        <li>
            <a b:behavior="focusobject" b:focusgroup="false" b:focusitem="true">Seven
Sumarai</a>
        </li>
        <li>
            <a b:behavior="focusobject" b:focusgroup="false" b:focusitem="true">The Huducker
Proxy</a>
        </li>
        <li>
            <a b:behavior="focusobject" b:focusgroup="false" b:focusitem="true">Life is
Beautiful</a>
        </li>
    </ul>
</div>
<div b:behavior="focusobject" b:focusgroup="true">Authors
    <ul>
        <li>
            <a b:behavior="focusobject" b:focusgroup="false" b:focusitem="true">Paul
Auster</a>
        </li>
        <li>
            <a b:behavior="focusobject" b:focusgroup="false" b:focusitem="true">Henry
Miller</a>
        </li>
        <li>
            <a b:behavior="focusobject" b:focusgroup="false" b:focusitem="true">Joseph
Conrad</a>
        </li>
    </ul>

```

```
</div>
```

## inactive

The `inactive` event takes place on an element when it, or a child node, loses focus.

### Implementation

In the following example, when you click within the `div` elements; the active elements are displayed in green and inactive elements in red:

```
<s:behavior b:name="showactiveelements">
    <s:event b:on="construct">
        <s:setstyle b:padding="10px" b:border="solid 1px black" />
    </s:event>
    <s:event b:on="active">
        <s:setstyle b:background-color="green" />
    </s:event>
    <s:event b:on="inactive">
        <s:setstyle b:background-color="red" />
    </s:event>
</s:behavior>
<div b:behavior="showactiveelements" b:focusroot="true" b:focusitem="true">FocusRoot
    <div b:behavior="showactiveelements" b:focusgroup="true" b:focusitem="true">Focusgroup 2
        <div b:behavior="showactiveelements" b:focusitem="true">Focusitem 2.1</div>
        <div b:behavior="showactiveelements" b:focusitem="true">Focusitem 2.2</div>
        <div b:behavior="showactiveelements" b:focusitem="true">Focusitem 2.3</div>
        <div b:behavior="showactiveelements" b:focusitem="true">Focusitem 2.4</div>
    </div>
</div>
```

## keydown

Takes place when the user depresses a key.

### Implementation

In the following example, a message is generated in the Runtime Tracer of the Backbase Developer Tool when a key is pressed in the input field:

```
<input type="input">
    <s:event b:on="keydown">
        <s:task b:action="msg" b:level="custom" b:value="keydown event triggered" />
    </s:event>
</input>
```

## keyup

Takes place when the user releases a key from its depressed position..

### Implementation

In the following example, a message is generated in the Runtime Tracer of the Backbase Developer Tool when a key that has been pressed is released in the input field:

```
<input type="input">
    <s:event b:on="keyup">
        <s:task b:action="msg" b:level="custom" b:value="keyup event triggered" />
    </s:event>
</input>
```

## load

The `load` event occurs when a document is loaded and parsed by the BPC. It can only be placed on BXML root tags (i.e. the tag where the Backbase namespaces are declared).

## mousedeeper

The `mousedeeper` event occurs when the user moves the mouse pointer from a parent element to a child element. The event is triggered on the parent element.

### Implementation

In the following example, the background color of the `div` elements changes according to the mouse event that is triggered on that element:

1. Entering Level 1, triggers the event `mouseenter` on Level 1.
2. Entering Level 2, triggers the event `mouseenter` on Level 2 and `mousedeeper` on Level 1.
3. Entering Level 3, triggers the event `mouseenter` on Level 3 and `mousedeeper` on Level 2.
4. Exiting Level 3, triggers the event `mouseleave` on Level 3 and `mousereenter` on Level 2.
5. Exiting Level 2, triggers the event `mouseleave` on Level 3 and `mousereenter` on Level 2.

```
<s:behavior b:name="mouseevents">
    <s:event b:on="construct">
        <s:setstyle b:padding="10px" b:border="solid 1px" />
    </s:event>
    <s:event b:on="mousedeeper">
        <s:setstyle b:background-color="red" />
    </s:event>
    <s:event b:on="mouseenter">
        <s:setstyle b:background-color="white" />
    </s:event>
    <s:event b:on="mouseleave">
        <s:setstyle b:background-color="yellow" />
    </s:event>
    <s:event b:on="mousereenter">
        <s:setstyle b:background-color="green" />
    </s:event>
</s:behavior>
<div b:behavior="mouseevents" style="background-color: #ECF5FF; width:400px">Level 1
    <div b:behavior="mouseevents" style="background-color: #A8D1FF">Level 2
        <div b:behavior="mouseevents" style="background-color: #5BA9FF">Level 3</div>
    </div>
</div>
```

## mousedown

The `mousedown` event occurs when the user presses the left button of the mouse pointer down.

### Implementation

In the following example, when you depress the left mouse button the `mousedown` event is triggered; releasing the button triggers the `mouseup` event. Event handlers are defined within a behavior that change the background and text color when these events are triggered.

```
<s:behavior b:name="mousedownandup">
    <s:event b:on="mousedown">
        <s:task b:action="set" b:target="//b:box/@b:style" b:value="background-color: red;
color:white" />
    </s:event>
    <s:event b:on="mouseup">
        <s:task b:action="set" b:target="//b:box/@b:style" b:value="background-color: green;
color:white" />
    </s:event>
</s:behavior>
```

```
<b:box b:behavior="mousedownandup" b:style="background-color: #CCFF00">           "Scarface
developed a cult following among younger audiences."</b:box>
```

## mouseenter

The `mouseenter` event occurs when the user moves the mouse pointer from a parent element to a sibling or child element. The event is triggered on the sibling or child element.

### Implementation

In the following example, the background color of the `div` elements changes according to the mouse event that is triggered on that element:

1. Entering Level 1, triggers the event `mouseenter` on Level 1.
2. Entering Level 2, triggers the event `mouseenter` on Level 2 and `mousedeeper` on Level 1.
3. Entering Level 3, triggers the event `mouseenter` on Level 3 and `mousedeeper` on Level 2.
4. Exiting Level 3, triggers the event `mouseleave` on Level 3 and `mousereenter` on Level 2.
5. Exiting Level 2, triggers the event `mouseleave` on Level 3 and `mousereenter` on Level 2.

```
<s:behavior b:name="mouseevents">
    <s:event b:on="construct">
        <s:setstyle b:padding="10px" b:border="solid 1px" />
    </s:event>
    <s:event b:on="mousedeeper">
        <s:setstyle b:background-color="red" />
    </s:event>
    <s:event b:on="mouseenter">
        <s:setstyle b:background-color="white" />
    </s:event>
    <s:event b:on="mouseleave">
        <s:setstyle b:background-color="yellow" />
    </s:event>
    <s:event b:on="mousereenter">
        <s:setstyle b:background-color="green" />
    </s:event>
</s:behavior>
<div b:behavior="mouseevents" style="background-color: #ECF5FF; width:400px">Level 1
    <div b:behavior="mouseevents" style="background-color: #A8D1FF">Level 2
        <div b:behavior="mouseevents" style="background-color: #5BA9FF">Level 3</div>
    </div>
</div>
```

## mouseleave

The `mouseleave` event occurs when the user moves the mouse pointer from a child element to a parent element. The event is triggered on the child element.

### Implementation

In the following example, the background color of the `div` elements changes according to the mouse event that is triggered on that element:

1. Entering Level 1, triggers the event `mouseenter` on Level 1.
2. Entering Level 2, triggers the event `mouseenter` on Level 2 and `mousedeeper` on Level 1.
3. Entering Level 3, triggers the event `mouseenter` on Level 3 and `mousedeeper` on Level 2.
4. Exiting Level 3, triggers the event `mouseleave` on Level 3 and `mousereenter` on Level 2.

## Level 2.

5. Exiting Level 2, triggers the event `mouseleave` on Level 3 and `mousereenter` on Level 2.

```
<s:behavior b:name="mouseevents">
  <s:event b:on="construct">
    <s:setstyle b:padding="10px" b:border="solid 1px" />
  </s:event>
  <s:event b:on="mousedeeper">
    <s:setstyle b:background-color="red" />
  </s:event>
  <s:event b:on="mouseenter">
    <s:setstyle b:background-color="white" />
  </s:event>
  <s:event b:on="mouseleave">
    <s:setstyle b:background-color="yellow" />
  </s:event>
  <s:event b:on="mousereenter">
    <s:setstyle b:background-color="green" />
  </s:event>
</s:behavior>
<div b:behavior="mouseevents" style="background-color: #ECF5FF; width:400px">Level 1
  <div b:behavior="mouseevents" style="background-color: #A8D1FF">Level 2
    <div b:behavior="mouseevents" style="background-color: #5BA9FF">Level 3</div>
  </div>
</div>
```

## mousereenter

The `mousereenter` event occurs when the user moves the mouse pointer from a child element to parent element. The event is triggered on the parent element.

### Implementation

In the following example, the background color of the `div` elements changes according to the mouse event that is triggered on that element:

1. Entering Level 1, triggers the event `mouseenter` on Level 1.
2. Entering Level 2, triggers the event `mouseenter` on Level 2 and `mousedeeper` on Level 1.
3. Entering Level 3, triggers the event `mouseenter` on Level 3 and `mousedeeper` on Level 2.
4. Exiting Level 3, triggers the event `mouseleave` on Level 3 and `mousereenter` on Level 2.
5. Exiting Level 2, triggers the event `mouseleave` on Level 3 and `mousereenter` on Level 2.

```
<s:behavior b:name="mouseevents">
  <s:event b:on="construct">
    <s:setstyle b:padding="10px" b:border="solid 1px" />
  </s:event>
  <s:event b:on="mousedeeper">
    <s:setstyle b:background-color="red" />
  </s:event>
  <s:event b:on="mouseenter">
    <s:setstyle b:background-color="white" />
  </s:event>
  <s:event b:on="mouseleave">
    <s:setstyle b:background-color="yellow" />
  </s:event>
  <s:event b:on="mousereenter">
    <s:setstyle b:background-color="green" />
  </s:event>
</s:behavior>
<div b:behavior="mouseevents" style="background-color: #ECF5FF; width:400px">Level 1
  <div b:behavior="mouseevents" style="background-color: #A8D1FF">Level 2
```

```
<div b:behavior="mouseevents" style="background-color: #5BA9FF">Level 3</div>
</div>
</div>
```

## mouseup

The `mouseup` event occurs when the user releases the mouse pointer's left button.

### Implementation

In the following example, when you depress the left mouse button the `mousedown` event is triggered; releasing the button triggers the `mouseup` event. Event handlers are defined within a behavior that change the background and text color when these events are triggered.

```
<s:behavior b:name="mousedownandup">
    <s:event b:on="mousedown">
        <s:task b:action="set" b:target="//b:box/@b:style" b:value="background-color: red;
color:white" />
    </s:event>
    <s:event b:on="mouseup">
        <s:task b:action="set" b:target="//b:box/@b:style" b:value="background-color: green;
color:white" />
    </s:event>
</s:behavior>
<b:box b:behavior="mousedownandup" b:style="background-color: #CCFF00">          "Scarface
developed a cult following among younger audiences."</b:box>
```

## open

The `open` event occurs when a node in a treelist or menu is opened.

## receive

The `receive` event occurs when an element receives an element as a result of a `copy`, `move`, or `render` action.

The following variables are available for this action (`_source` and `_sourceIndex` do not apply to `copy` and `render` actions):

- `_source` - contains the parent element of the element on which the action is being performed
- `_element` - contains the element on which the action is being performed
- `_target` - contains the element which is the target of the action
- `_sourceIndex` - contains the child element index of the element on which the action is being performed relative to its parent; that is, whether it was the first, second, third, fourth, and so on child of the parent

### Implementation

In the following example, you can click an item in the list to move it from one list to another. When you do so, a `receive` event is triggered; an event handler is defined for `receive` that activates an alert box to inform you of the element that was moved, from which element and to where (using the available `receive` variables):

```
<div id="container">
    <p>Click the articles to move them to your shopping list...</p>
    <b:box b:style="width: 400px; height: 100px">
        <p>Product List:</p>
        <ul id="products list">
            <li b:action="move" b:source=".." b:destination="id('container')//ul except ...
b:mode="aslastchild">Cheese</li>
            <li b:action="move" b:source=".." b:destination="id('container')//ul except ..."
```

```

b:mode="aslastchild">Eggs</li>
    <li b:action="move" b:source=". " b:destination="id('container')//ul except .. "
b:mode="aslastchild">Milk</li>
    </ul>
</b:box>
<br />
<b:box b:style="width: 400px; height: 150px">
    <p>My Shopping List:</p>
    <ul id="shopping list">
        <s:event b:on="receive">
            <s:task b:action="alert" b:value="{concat ('The element ', $_element, ' was
moved from the ', $_source/@id, ' to the ', $_target/@id)}" />
        </s:event>
    </ul>
</b:box>
</div>
```

## remove

The `remove` event occurs when the element is destroyed by a `remove` command.

## resize

The `resize` event occurs on an element when it is resized (an element is resizable when the `b:resize` attribute is set).

### Implementation

In the following example, when you resize the element the `resize` event handler uses the `setcookie` command to set cookies for the element's style height and width properties (`style::` is a Backbase XPath axis for targetting individual style property values). A `construct` event handler is also defined that first tests whether the cookies exist (using the Backbase XPath function `cookie()`), and if so it sets the width and height properties to the values stored in the cookie:

```

<div b:resize="all" style="position: absolute; left: 10px; top: 10px; border: solid 2px red;
width: 200px; height: 200px;">
    <p>This element is resizable.</p>
    <p>When you resize the element, it sets a cookie.</p>
    <p>Refresh the page to see element height and width values stored in the cookie being
used to reconstruct the element to its last used dimensions.</p>
    <s:event b:on="resize">
        <s:task b:action="setcookie" b:name="div-height" b:value="{style::height}" b:days="3"
/>
        <s:task b:action="setcookie" b:name="div-width" b:value="{style::width}" b:days="3"
/>
    </s:event>
    <s:event b:on="construct">
        <s:task b:test="cookie('div-height')" b:action="set" b:value="{cookie('div-height')}"
b:target="style::height" />
        <s:task b:test="cookie('div-width')" b:action="set" b:value="{cookie('div-width')}"
b:target="style::width" />
    </s:event>
</div>
```

## resizewindow

The `resizewindow` event occurs when the browser window is resized. This event is triggered on the `body` tag.

## rmbclick

The `rmbclick` event occurs when a user clicks on an element with the right mouse button.

## rmbdown

The `rmbdown` event occurs when the user presses the right button of the mouse pointer down.

### Implementation

In the following example, when you depress the right mouse button on the `div` element, the `rmbdown` event is triggered; an event handler is defined within a behavior that selects and positions a context-sensitive pop-up menu (note the `b:bubble="false"` attribute suppresses the browser's event handler for the `rmbdown` event):

```
<s:behavior b:name="rmbdown">
  <s:event b:on="rmbdown" b:bubble="false">
    <s:task b:action="select" b:target="id('contextmenu')" />
    <s:task b:action="position" b:type="place" b:target="id('contextmenu')"
b:destination=".." b:position="at-pointer" />
  </s:event>
</s:behavior>
<div b:behavior="rmbdown">Right-click here to see context menu...
  <b:contextmenu id="contextmenu">
    <b:contextmenurow b:label="Open" />
    <b:contextmenurow b:label="Close" />
    <b:contextmenurow b:label="Exit" />
  </b:contextmenu>
</div>
```

## rmbup

The `rmbup` event occurs when the user releases the mouse pointer's right button.

### Implementation

In the following example, when you depress the right mouse button the `rmbdown` event is triggered; releasing the button triggers the `rmbup` event. Event handlers are defined within a behavior that change the background and text color when these events are triggered.

```
<s:behavior b:name="rmbmousedownandup">
  <s:event b:on="rmbdown">
    <s:task b:action="set" b:target=".@b:style" b:value="background-color: red;
color:white" />
  </s:event>
  <s:event b:on="rmbup">
    <s:task b:action="set" b:target=".@b:style" b:value="background-color: green;
color:white" />
  </s:event>
</s:behavior>
<b:box b:behavior="rmbmousedownandup" b:style="background-color: #CCFF00">      "Scarface
developed a cult following among younger audiences."</b:box>
```

## scrollwindow

The `scrollwindow` event occurs when the browser window is scrolled. This event is triggered on the `body` tag.

## select

The `select` event occurs when the element is put in the `selected` state.

### Implementation

In the following example, the behavior contains three event handlers: when the `div` element is clicked, a `select-deselect` action is triggered that toggles the element's state

from selected to deselected, and triggers respectively either the `select` event handler that shows the child `div` element, or the `deselect` event handler that hides the child `div` element.

```
<style type="text/css">.myClass { font-size: 16px; line-height: 22px; border: 1px solid #CCC; width: 200px }</style>
<s:behavior b:name="openChildren">
  <s:event b:on="command">
    <s:task b:action="select-deselect" b:target=". " />
  </s:event>
  <s:event b:on="select">
    <s:task b:action="show" b:target="div" />
  </s:event>
  <s:event b:on="deselect">
    <s:task b:action="hide" b:target="div" />
  </s:event>
</s:behavior>
<div b:behavior="openChildren" class="myClass">Seven Samurai&gt;&gt;
  <div style="font-size: 10px; display: none">A film by....</div>
</div>
```

## slide

The `slide` event occurs when the user moves the thumb of a slider control.

### Implementation

In the following example, when you move the slider the `slide` event handler in the behavior is triggered which reads out the current value of the `b:value` attribute and inserts it in the text node of the `span id="readVariableValue"` element:

```
<s:behavior b:name="slide">
  <s:event b:on="slide">
    <s:task b:action="set" b:target="id('currentvalue')/text()" b:value="@b:value" />
  </s:event>
</s:behavior>
<form>
  <div style="position: absolute; top: 20px; left: 20px">
    <span id="currentvalue">50</span>
    <b:slider b:behavior="slide" id="sliderHorizontal" b:value="50"
b:orientation="horizontal" b:start="0" b:end="100" b:step="2" />
    <br />
    <input type="text" b:connect="id('sliderHorizontal')" value="" />
  </div>
</form>
```

## submit

The `submit` event occurs when the user submits a form. It is within the handler of this submit event that the `send` command should be issued that finally submits the form, once all of the validation conditions have been satisfied.

## Backbase JavaScript Functions

### bpc.addClass()

Syntax: `void bpc.addClass(sValue, oElm)`

Adds a class name on the given object. This function does the same as the `addclass` command.

The function accepts two arguments: a string containing the class name to add, and the object element to which the named class is added.

### bpc.checkEventBlock()

Syntax: `bpc.checkEventBlock(oElm)`

This function checks to see if the input is blocked on the given element according to BXML rules.

All real JavaScript events (outside the scope of BPC) simply ignore the BXML rules that enable or disable input events on elements. This function can be used to check if your custom javascript events should be blocked according to BXML rules.

### Implementation

```
<script type="text/javascript">//<![CDATA[
    function showAlert(oElm) {
        if (bpc.checkEventBlock(oElm)) {
            alert('Not blocked');
        } else {
            alert('Blocked')
        }
    }
//]]>

</script>

<b:xhtml b:disabled="true">
    <div onclick="showAlert(this)">This element is blocked in BXML by its parent</div>
</b:xhtml>
<b:xhtml>
    <div onclick="showAlert(this)">This element is not blocked in BXML</div>
</b:xhtml>
```

### bpc.execute()

Syntax: `void bpc.execute(sCommands, oElm)`

Executes an XML string. This function does the same as the `s:execute` tag

The function accepts two arguments: a string of XML that constitutes a valid set of instructions to execute, and an element to use as the context node for the tasks. The element is optional, if the instruction set targets an element.

The instruction set can consist of anything that is allowed inside an `s:event` or `s:execute` tag. Remember that not all HTML elements are in the BXML space: passing an HTML element instructs the BPC to look for the first parent of that HTML element that is inside the BXML tree.

## Implementation

In the following example, clicking the link removes the `b:box` and triggers an alert box. Note that the id of the `b:box` is targeted, and that the single quotes are escaped using `\'`.

```
<b:box id="box" b:resize="all">
  <div>Resizable box</div>
</b:box>
<a>Execute
  <s:event b:on="command">
    <s:script>
      <!-- bpc.execute('<s:task b:action="remove" b:target="id(\\'box\\\' )" /><s:task
b:action="alert" b:value="Box removed!" />'); -->
    </s:script>
  </s:event>
</a>
```

In the following example, clicking the link removes itself, using the `_current` JavaScript variable to select it as the context node for the instruction set.

```
<a>Execute
  <s:event b:on="command">
    <s:script>
      <!-- bpc.execute('<s:task b:action="remove" b:target="id(\\'box\\\' )" />',
_current'); -->
    </s:script>
  </s:event>
</a>
```

## **bpc.focus()**

Syntax: `void bpc.focus(oElm)`

Puts focus on the passed object element. This function does the same as the `focus` command.

## **bpc.forceDisplay()**

Syntax: `bpc.forceDisplay(oElm)`

Forces an element to be visible by setting all its parent style to `display:inherit`. It is used when you want to calculate a size of a element which is hidden because of its parent's value, for example in a `b:tabbox` or `b:deck`.

## **bpc.getAttribute()**

Syntax: `bpc.getAttribute(sName, oElm)`

Gets a BXML attribute.

The function accepts two arguments: a string containing the attribute name (`sName`), and the object element (`oElm`) in which the attribute is defined.

## Implementation

In the following example, clicking the first link sets the BXML attribute `b:resize` to `all` on the `b:box` element, and then gets the value and displays it in an alert box. Clicking the second link removes the attribute.

```
<a>Add resize
  <s:event b:on="command">
    <s:script>
      <!-- var elm = document.getElementById('box');
bpc.setAttribute('b:resize', 'all', elm); var rv = bpc.getAttribute('b:resize', elm);
alert('resize is: ' + rv);-->
    </s:script>
```

```

        </s:event>
    </a>
<b:box id="box">
    <div>Resizable box</div>
</b:box>
<a>Remove resize
    <s:event b:on="command">
        <s:script>
            <!-- bpc.removeBXMLAttribute('b:resize', document.getElementById('box')) ;
alert('resize is disabled');-->
</s:script>
    </s:event>
</a>

```

## **bpc.getCookie()**

Syntax: `bpc.getCookie(sName)`

Gets a cookie from the users computer.

This function accepts one argument: the name of the cookie to get.

### Implementation

```

<script type="text/javascript">//<![CDATA[
    function myfunc() {

        bpc.setCookie('js-box-width','300', 7);
        bpc.setCookie('js-box-height','300',7);
        var width = bpc.getCookie('js-box-width');
        var height = bpc.getCookie('js-box-height');
    }]]>

}

</script>

```

Note: You can execute JavaScript functions using the `js` command, or the `s:script` element.

## **bpc.getNextElementByNodeType()**

Syntax: `bpc.getNextElementByNodeType(iNodeType, oElm)`

The element that is passed is checked to see whether it matches the given node type. If it matches, that element is returned; if it does not, the next element is checked, until it finds a match or if there are no next sibling elements.

The element expects two arguments: the nodetype to check as an integer and the element to start the check with. The most important nodetype values are: 1 for elements, 2 for attributes and 3 for text nodes.

## **bpc.getNextElementByTagName()**

Syntax: `bpc.getNextElementByTagName(sTagName, oElm)`

The element that is passed is checked to see whether it matches the given tag name. If it matches, that element is returned; if it does not, the next element is checked, until it finds a match or if there are no next sibling elements.

## **bpc.getPreviousElementByNodeType()**

Syntax: `bpc.getPreviousElementByNodeType(iNodeType, oElm)`

The element that is passed is checked to see whether it matches the given node type. If it matches, that element is returned; if it does not, the previous element is checked, until it finds a match or if there are no previous sibling elements.

The element expects two arguments: the nodetype to check as an integer and the element to start the check with. The most important nodetype values are: 1 for elements, 2 for attributes and 3 for text nodes.

## bpc.getPreviousElementByTagName()

Syntax: `bpc.getPreviousElementByTagName(sTagName, oElm)`

The element that is passed is checked to see whether it matches the given tag name. If it matches, that element is returned; if it does not, the previous element is checked, until it finds a match or if there are no previous sibling elements.

## bpc.getSize()

Syntax: `bpc.getSize(oElm, sMode)`

Gets the size of an attribute.

This function accepts two arguments: the `oElm` parameter points to an element in the HTML space, the `sMode` parameter is a string that determines what box of the element to measure and can have the following values: `border` (the default), `margin`, `padding`, or `content`.

### Implementation

In the following example, the `oElm` parameter points to an element in the HTML space. The return value is a hash array with indices 'x', 'y', 'w' and 'h'.

```
<a>Show position & dimensions
<s:event b:on="command">
    <s:script>
        <!-- var rv = bpc.getSize(document.getElementById('yyy'), 'padding'); alert('x:' +
+ rv['x'] + ' y:' + rv['y'] + ' w:' + rv['w'] + ' h:' + rv['h']);-->
    </s:script>
</s:event>
</a>
<div b:resize="all" id="yyy" b:drag="xxx" style="position: absolute; height: 100px; width:
200px; border: 1px dashed green; background: #9F9; padding: 0.5em; margin: 1em;">Drag or
resize me</div>
```

Note: You can execute JavaScript functions using the `js` command, or the `s:script` element.

## bpc.getXpathVar()

Syntax: `void bpc.getXpathVar(sName)`

Gets an XPath variable. Note that this function has been deprecated: use the `_vars` array inside the `s:script` tag to get XPath variables.

The function accepts one argument: the name of the variable to get.

## bpc.move()

Syntax: `bpc.move(sXML, sMode, sTrg, oElm)`

Inserts an HTML fragment into a specified location. (This function is deprecated, the new function is called `bpc.render`)

In the following example, the value `greeting` in the move function is inserted after the

text input type whose ID is destination:

## Implementation

The following is an example of defining a **bp**c.move function:

```
<span>Enter Text
  <input id="destination" type="text">
    <s:event b:on="command">
      <s:script>
        <!-- var greeting = 'hello!!!!'; bp
          .move('<span>' + greeting + '</span>',
            'after' , 'id("destination")') -->
      </s:script>
    </s:event>
  </input>
</span>
```

Note: You can execute JavaScript functions using the **js** command, or the **s:script** element.

## **bp**c.place()

Syntax: **void** **bp**c.place(**oSelect**, **oContext**, **sMode**)

Changes the position of an element on the screen. This function is the same as the **position** command used with **b:type="place"**.

The function accepts three arguments: the element to position, the element to use as a context node to position against, and the mode to use for placement (optional). The same modes that can be used with the **position** command in combination with the **b:type="place"** attribute specification can also be used here.

## Implementation

```
<script type="text/javascript">
  function moveMe(){
    bp
    .place(document.getElementById('place'),document.body,'at-pointer');
    setTimeout(moveMe,100);
  }</script>
```

```
<span id="place" style="position:absolute;" onclick="moveMe();">Start snap to mouse!
  <input type="text" />
</span>
```

Note: You can execute JavaScript functions using the **js** command, or the **s:script** element.

## **bp**c.removeBXMLAttribute()

Syntax: **bp**c.removeBXMLAttribute(**sName**, **oElm**)

Removes a BXML attribute.

The function accepts two arguments: a string containing the attribute name (**sName**), and the object element (**oElm**) in which the attribute is defined.

## Implementation

In the following example, clicking the first link sets the BXML attribute **b:resize** to **all** on the **b:box** element, and then gets the value and displays it in an alert box. Clicking the second link removes the attribute.

```
<a>Add resize
  <s:event b:on="command">
    <s:script>
```

```

<!-- var elm = document.getElementById('box');
bpc.setBXMLAttribute('b:resize', 'all', elm); var rv = bpc.getBXMLAttribute('b:resize', elm);
alert('resize is: ' + rv);-->
</s:script>
</s:event>
</a>
<b:box id="box">
  <div>Resizable box</div>
</b:box>
<a>Remove resize
  <s:event b:on="command">
    <s:script>
      <!-- bpc.removeBXMLAttribute('b:resize', document.getElementById('box')) ;
alert('resize is disabled');-->
    </s:script>
  </s:event>
</a>

```

## bpc.removeClass()

Syntax: `void bpc.removeClass(sValue, oElm)`

Removes a class name from the given object. This function does the same as the `removeclass` command.

The function accepts two arguments: a string containing the class name to remove, and the object element which the named class is removed from.

## bpc.render()

Syntax: `void bpc.render(sXML, sMode, sTrg, oElm)`

Places an XML fragment into the BXML tree. This function does the same as the `s:render` tag.

This function accepts four arguments: a string containing valid XML (so it should have a single root node), a string denoting the insertion mode (`asfirstchild`, `aslastchild`, `replacechildren`, `replace`, `before` and `after`), a string with a valid XPath expression and an object denoting the context node to use. `bpc.render()` returns void.

### Implementation

```

<div id="mytargetid">Backbase Rich Internet Applications</div>

<script type="text/javascript">
  //
  bpc.render('<p>This text replaces the text that was originally here!</p>', 
'replace', 'id("mytargetid")');
//</script>

```

Note: You can execute JavaScript functions using the `js` command, or the `s:script` element.

## bpc.restoreDisplay()

Syntax: `bpc.restoreDisplay(oElm)`

Restores the state of an element to the original after a `bpc.forceDisplay`.

## bpc.setBXMLAttribute()

Syntax: `bpc.setBXMLAttribute(sName, oElm)`

Sets a BXML attribute.

The function accepts two arguments: a string containing the attribute name (`sName`), and the object element (`oElm`) in which the attribute is defined.

## Implementation

In the following example, clicking the first link sets the BXML attribute `b:resize` to `all` on the `b:box` element, and then gets the value and displays it in an alert box. Clicking the second link removes the attribute.

```
<a>Add resize
  <s:event b:on="command">
    <s:script>
      <!-- var elm = document.getElementById('box');
bpc.setBXMLAttribute('b:resize','all', elm); var rv = bpc.getBXMLAttribute('b:resize', elm);
alert('resize is: ' + rv);-->
    </s:script>
  </s:event>
</a>
<b:box id="box">
  <div>Resizable box</div>
</b:box>
<a>Remove resize
  <s:event b:on="command">
    <s:script>
      <!-- bpc.removeBXMLAttribute('b:resize', document.getElementById('box'));
alert('resize is disabled');-->
    </s:script>
  </s:event>
</a>
```

## **bpc.setCookie()**

Syntax: `bpc.setCookie(sName, sValue, iDays)`

Sets a cookie on the users computer.

The function accepts three arguments: the name of the cookie, the value to put in the cookie, and the number of days the cookie is stored before expiring.

## Implementation

```
<script type="text/javascript">//<![CDATA[
  function myfunc() {
    bpc.setCookie('js-box-width','300', 7);
    bpc.setCookie('js-box-height','300',7);
    var width = bpc.getCookie('js-box-width');
    var height = bpc.getCookie('js-box-height');
    alert('Set height=' + height + '\nSet width=' + width);
  }
  myfunc();
//]]>

</script>
```

## **bpc.setVariable()**

Syntax: `bpc.setVariable(sName, [sValue], oElm)`

Gets and sets a `tag` variable, if you specify the `oElm` argument, or `global` variable if the `oElm` argument is omitted.

This function can only be used inside an `s:script` tag; this tag sets the required context

to execute this function. The context node is the node in which the `s:script` tag is executed. This function accepts three arguments: the name of the variable, the value to assign it, and optionally the element in which it is defined.

## Implementation

```
<div id="my-div" b:myatt="Hello world!">
  <s:event b:on="construct">
    <s:variable b:name="myTag" b:scope="global" b:select="@b:myatt" />
  </s:event>
</div>
<div id="my-div2" b:action="alert" b:value="{{$myTag}}>Click to view $myTag variable</div>
<a>Click here to modify $myTag variable from JavaScript
  <s:event b:on="command">
    <s:script>
      <!-- bpc.setVariable('myTag', [document.getElementById('my-div2')]);-->
    </s:script>
  </s:event>
</a>
```

## bpc.setXpathVar()

Syntax: `void bpc.setXpathVar(sName, sValue)`

Sets an XPath variable (this function has been deprecated).

The function accepts two arguments: the name of the variable to get and the string value to set. The new and better way of setting Xpath variables is to use the `bpc.setVariable` function inside the `s:script` tag.

## bpc.task()

Syntax: `bpc.task(aParam, oElm)`

Executes the given array as a BXML task with the given context node.

This function accepts two arguments: an array of attribute/value pairs, and an element to use as the context node. Every odd numbered record in the array is the name of the attribute, every event numbered record in the array is the value of the attribute.

## Implementation

```
<s:execute>
  <s:script>
    <!-- bpc.task(['b:action', 'alert', 'b:value', 'Hello world!'], _current)-->
  </s:script>
</s:execute>
```

## bpc.toString()

Syntax: `bpc.toString(sXPath, oContext)`

Returns a string that contains the full XML syntax of the targeted element.

This function accepts two arguments: an XPath expression, and the `context` (optional).

## Implementation

The following example results in an alert `<div class="now"> The Time is Now! </div>`.

```
<div id="test">
  <div class="now">The Time is Now!</div>
  <div>More content</div>
</div>
```

```
<script type="text/javascript">
    var oContext = document.getElementById('test');
    var sText = bpc.toString( "./div[@class='now']", oContext);
    alert(sText);</script>
```

---

Note: You can execute JavaScript functions using the `js` command, or the `s:script` element.

## bpc.trigger()

Syntax: `void bpc.trigger(sEvent, oElm)`

Triggers an event on the given element. This function does the same as the `trigger` command.

The function accepts two arguments: a string with the BXML event to trigger and an object to trigger the event on. Remember that not all HTML elements are in the BXML space: passing an HTML element instructs the BPC to look for the first parent of that HTML element that is inside the BXML tree.

## bpc.xpath()

Syntax: `bpc.xpath(sXPath, oElm, bCreateNonExistingAttributes)`

Evaluates an XPath statement and returns the result. The function returns either a value, or an array of HTML elements. Note that if an XPath is expected to return only one element, it will still return an array of elements. If the result is a node-set, the node-set is returned as an array of HTML DOM elements.

This function accepts two arguments: a string containing an XPath statement, and an object reference to an HTML element. The HTML element is used as a starting point (context node) for the XPath to execute.

The following example puts a border around some of the span elements.

## Implementation

```
<div id="test" bla="not successful">
    <s:event b:on="command">
        <s:script><!-- aNodes = bpc.xpath("descendant::span[text()='border']");
_current); for (var i = 0; i< aNodes.length; i++) {
aNodes[i].style.border ='1px solid green'; -->
        </s:script>
    </s:event>
    <div>Click me to set borders</div>
    <span>border</span>
    <span>no border</span>
    <span>border</span>
    <span>border</span>
</div>
```

---

Note: You can execute JavaScript functions using the `js` command, or the `s:script` element.

## XPath Axes

### ancestor::

Selects all ancestors (parent, grandparent, and so on, ) of the current node.

### ancestor-or-self::

Selects all ancestors (parent, grandparent, and so on) of the current node and the cur-

rent node itself.

### **attribute::**

Selects all attributes of the current node.

### **child::**

Selects all child elements of the current node.

### **descendant::**

Selects all descendants (children, grandchildren, and so on) of the current node.

### **descendant-or-self::**

Selects all descendants (children, grandchildren, and so on) of the current node and the current node itself.

### **following-sibling::**

Selects all siblings after the current node. The Backbase shorthand for following-sibling is `~+`.

### **object::**

Selects the values of properties of selected nodes (Backbase custom Axis). The property of an `object` can also be an object with its own properties, so it is valid to write for example `/div/span/object::parentNode.width`

## Implementation

```
<div id="myid">
    <s:event b:on="command">
        <s:task b:action="alert" b:value="{object::mycustomvalue}" />
    </s:event>
    Click Me!
</div>
<script type="text/javascript">      var elm = document.getElementById('myid');
elm.mycustomvalue = 'Hello World!';</script>
```

### **parent::**

Selects the parent of the current node.

### **preceding-sibling::**

Selects all siblings before the current node. The Backbase shorthand for preceding-sibling is `~-`.

### **self::**

Selects the current node.

### **style::**

Accesses and updates style properties using their CSS names (Backbase custom Axis).

## Implementation

```
<div style="color: black; font-size: xx-large">
  <s:event b:on="command">
    <s:task b:action="set" b:target="style::color" b:value="red" />
    <s:task b:action="set" b:target="style::font-size" b:value="small" />
  </s:event>
  Click Me!
</div>
```

## XPath Functions

### **abs()**

`abs(num)` returns the absolute value of the argument.

For example, `abs(3.14)` results in 3.14, `abs(-3.14)` results in 3.14

### **avg()**

`avg((arg,arg,...))` returns the average of the argument values.

For example: `avg((1,2,3))` results in 2

### **boolean()**

A Boolean function that converts its argument to a Boolean value. It uses the following rules:

- Zero and NaN are converted to false. All other numbers are true.
- Empty node sets are converted to false. Non-empty node sets are converted to true.
- Empty Strings are converted to false. Non-empty Strings are considered true.

### **bxml()**

`bxml()` is a Backbase custom function that instructs the BPC to move from the HTML space to the BXML space (`b:bxml` elements within `b:xhtml` elements).

#### Implementation

In the following example, the second link targets elements in the HTML space using the `html()` walker, and once in the HTML space it targets elements within `b:bxml` element using the `bxml()` walker:

```
<a b:action="alert" b:value="{following::b:xhtml/html()/div/span}">To HTML Space</a>
<span></span>
<a b:action="alert" b:value="{following::b:xhtml/html()/bxml()/b:bxml/b:box}">To BXML Space
within HTML Space</a>
<b:xhtml>
  <div>
    <span>This element is in the HTML Space</span>
    <b:bxml>
      <b:box>This box is in the BXML Space</b:box>
    </b:bxml>
  </div>
</b:xhtml>
```

### **ceiling()**

`ceiling(num)` returns the smallest integer that is greater than the number argument.

For example: `ceiling(3.14)` results in 4

### **codepoints-to-string()**

`codepoints-to-string(int,int,...)` returns a string from a sequence of code points.

For example: `codepoints-to-string(84, 104, 233, 114, 232, 115, 101)` results in

'Thérèse'

## compare()

`compare(comp1,comp2)` or `compare(comp1,comp2,collation)` returns -1 if comp1 is less than comp2, 0 if comp1 is equal to comp2, or 1 if comp1 is greater than comp2 (according to the rules of the collation that is used).

For example: `compare('ghi', 'ghi')` results in 0

## concat()

`concat(string,string,...)` returns the concatenation of the strings.

For example: `concat('XPath ','is ','FUN!')` results in: 'XPath is FUN!'

## contains()

`contains(string1,string2)` returns true if string1 contains string2, otherwise it returns false

For example: `contains('XML','XM')` results in true

## cookie()

`cookie('cookieName')` is a Backbase custom function for getting a cookie.

### Implementation

In the following example, when you resize the box the `b:on="resize"` event handler sets cookies to store the height and width values of the `b:box`. The value is declared as a variable using curly brackets. The `b:on="construct"` event handler ensures that when the page is reloaded, the height and width properties are set using the cookie values:

```
<div b:resize="all" style="position: absolute; left: 10px; top: 10px; border: solid 2px red; width: 200px; height: 200px;">
    <p>This element is resizable.</p>
    <p>When you resize the element, it sets a cookie.</p>
    <p>Refresh the page to see element height and width values stored in the cookie being used to reconstruct the element to its last used dimensions.</p>
    <s:event b:on="resize">
        <s:task b:action="setcookie" b:name="div-height" b:value="{style::height}" b:days="3" />
        <s:task b:action="setcookie" b:name="div-width" b:value="{style::width}" b:days="3" />
    </s:event>
    <s:event b:on="construct">
        <s:task b:test="cookie('div-height')" b:action="set" b:value="{cookie('div-height')}" b:target="style::height" />
        <s:task b:test="cookie('div-width')" b:action="set" b:value="{cookie('div-width')}" b:target="style::width" />
    </s:event>
</div>
```

## count()

`count((item,item,...))` returns the number of nodes in the node set, which is passed as an argument.

## current()

Backbase custom function that returns the original context node (useful in predicates).

## current-date()

`current-date()` returns the current date (with timezone).

## current-dateTime()

`current-dateTime()` returns the current dateTime (with timezone).

## current-time()

`current-time()` returns the current time (with timezone).

## day-from-date()

`day-from-date(date)` returns an integer that represents the day in the localized value of the argument.

For example: `day-from-date(xs:date("2005-04-23"))` results in 23

## day-from-dateTime()

`day-from-dateTime(datetimetz)` returns an integer that represents the day component in the localized value of the argument.

For example: `day-from-dateTime(xs:dateTime("2005-01-10T12:30-04:10"))` results in 10

## days-from-duration()

`days-from-duration(datetimedur)` returns an integer that represents the days component in the canonical lexical representation of the value of the argument.

## declared()

`declared()` is a Backbase custom function to test whether a Backbase variable has been declared. Returns `true` if the variable has been declared, otherwise `false`.

### Implementation

```
<s:event b:on="construct">
  <s:variable b:name="myVar" b:scope="global" b:select="'Hi'" />
</s:event>
<a b:test="declared($myVar)" b:action="alert" b:value="Variable has been declared">Check
whether variable has been declared</a>
```

## deep-equal()

`deep-equal(param1,param2,collation)` returns true if param1 and param2 are deep-equal to each other, otherwise it returns false.

## distinct-values()

`distinct-values((item,item,...),collation)` returns only distinct (different) values.

For example: `distinct-values((1, 2, 3, 1, 2))` results in (1, 2, 3)

## empty()

`empty(item,item,...)` returns true if the value of the arguments IS an empty sequence, otherwise it returns false.

For example: `empty(remove(("ab", "cd"), 1))` results in false

## ends-with()

`ends-with(string1,string2)` returns true if string1 ends with string2, otherwise it returns false.

For example: `ends-with('XML','X')` results in false

## error()

`error(), error(error), error(error,description), error(error,description,error-object)` returns an error message.

For example:

```
error(fn:QName('http://example.com/test', 'err:toohigh'), 'Error: Price is too high')
```

results in

Returns `http://example.com/test#toohigh` and the string "Error: Price is too high" to the external processing environment

## escape-uri()

`escape-uri(stringURI,esc-res)` escapes paes returns

For example: `escape-uri("http://example.com/test#car", true())` results in "http%3A%2F%2Fexample.com%2Ftest#car"

For example: `escape-uri("http://example.com/test#car", false())` results in "http://example.com/test#car"

For example: `escape-uri ("http://example.com/~bébé", false())` results in "http://example.com/~b%C3%A9b%C3%A9"

## exactly-one()

`exactly-one(item,item,...)` returns the argument if it contains exactly one item, otherwise it raises an error.

## exists()

`exists(item,item,...)` returns true if the value of the arguments IS NOT an empty sequence, otherwise it returns false.

For example: `exists(remove(("ab"), 1))` results in false

## false()

`false()` returns (always) a Boolean value of false. It makes up for the lack of boolean literals in XPath.

For example: `false()` results in false

## floor()

`floor(num)` returns the largest integer that is not greater than the number argument.

For example: `floor(3.14)` results in 3

## hours-from-dateTime()

`hours-from-dateTime(datetime)` returns an integer that represents the hours component in the localized value of the argument.

For example: `hours-from-dateTime(xs:dateTime("2005-01-10T12:30-04:10"))` results in 12

## hours-from-duration()

`hours-from-duration(datetimedur)` returns an integer that represents the hours component in the canonical lexical representation of the value of the argument.

## hours-from-time()

`hours-from-time(time)` returns true if the value of the arguments IS NOT an empty sequence, otherwise it returns false.

For example: `hours-from-time(xs:time("10:22:00"))` results in 10

## html()

`html()` is a Backbase custom function that instructs the BPC to move from the BXML space to the HTML space. You can then target elements in the HTML space using standard XPath.

### Implementation

In the following example, clicking the button triggers an event handler that targets elements in the BXML and HTML Spaces to show and hide these elements.

```
<div style="background-color: #FE987E; width: 400px; height: 100px">These elements are in the
BXML space
    <b:button>Click here to Show-Hide elements in the BXML and HTML Space
        <s:event b:on="command">
            <s:task b:action="show-hide" b:target="id('html_space')/html()/div/span" />
            <s:task b:action="show-hide" b:target="id('bxml_space')/div[1]" />
        </s:event>
    </b:button>
    <div id="bxml_space">
        <div>Clicking the button toggles the display style property of this element in the
        BXML Space</div>
    </div>
</div>
<b:xhtml id="html_space">
    <div style="background-color: #80FFB3; width: 400px; height: 100px">These elements are in
    the HTML Space
        <span>Clicking the button toggles the display style property of this element in the
        HTML Space</span>
    </div>
</b:xhtml>
```

## id()

`id((string,string,...),node)` returns the node, whose id attribute matches the value given as its argument. Note that the argument can either be a string or an XPath statement, which can be cast to a String. When a space separated list of strings are given, an attempt is made to retrieve multiple nodes.

For example: `id('mainlayer')` selects the element whose `id` is '`mainlayer`'

## implicit-timezone()

`implicit-timezone()` returns the value of the implicit timezone.

## index-of()

`index-of((item,item,...),searchitem)` returns the positions within the sequence of

items that are equal to the searchitem argument.

For example: `index-of ((15, 40, 25, 40, 10), 40)` results in (2, 4)

For example: `index-of (( "a", "dog", "and", "a", "duck"), "a")` results in (1, 4)

For example: `index-of ((15, 40, 25, 40, 10), 18)` results in ()

## insert-before()

`insert-before((item,item,...),pos,inserts)` returns a new sequence constructed from the value of the item arguments - with the value of the inserts argument inserted in the position specified by the pos argument.

For example: `insert-before(("ab", "cd"), 0, "gh")` results in ("gh", "ab", "cd")

For example: `insert-before(("ab", "cd"), 1, "gh")` results in ("gh", "ab", "cd")

## last()

`last()` returns the size of current context node set.

For example: `//book[last()]` results in Selects the last book element

## local-name()

Returns the name of the current node or the first node in the specified node set - without the namespace prefix.

## local-name-from-QName()

## lower-case()

`lower-case(string)` converts the string argument to lower-case.

For example: `lower-case('The XML')` results in 'the xml'

## matches()

`matches(string,pattern)` returns true if the string argument matches the pattern, otherwise, it returns false.

For example: `matches("Merano", "ran")` results in true

You can also use an optional argument `flags`, which specifies one or more letters indicating options on how the matching is to be performed.

## max()

`max((arg,arg,...))` returns the argument that is greater than the others.

For example: `max((1,2,3))` results in 3

For example: `max(( 'a' , 'k' ))` results in 'k'

## min()

`min((arg,arg,...))` returns the argument that is less than the others.

For example: `min((1,2,3))` results in 1

For example: `min(( 'a' , 'k' ))` results in 'a'

## minutes-from-datetime()

`minutes-from-datetime(datetime)` returns an integer that represents the minutes com-

ponent in the localized value of the argument.

For example: `minutes-from-datetime(xs:dateTime("2005-01-10T12:30-04:10"))`  
results in 30

## minutes-from-duration()

`minutes-from-duration(datetimedur)` returns an integer that represents the minutes component in the canonical lexical representation of the value of the argument.

## minutes-from-time()

`minutes-from-time(time)` returns an integer that represents the minutes component in the localized value of the argument.

For example: `minutes-from-time(xs:time("10:22:00"))` results in 22

## month-from-date()

`month-from-date(date)` returns an integer that represents the month in the localized value of the argument.

For example: `month-from-date(xs:date("2005-04-23"))` results in 4

## month-from-datetime()

`month-from-datetime(datetime)` returns an integer that represents the month component in the localized value of the argument.

For example: `month-from-datetime(xs:dateTime("2005-01-10T12:30-04:10"))` results in 01

## months-from-duration()

`months-from-duration(datetimedur)` returns an integer that represents the months component in the canonical lexical representation of the value of the argument.

## name()

`name()`, `name(nodeset)` returns the name of the current node or the first node in the specified node set.

## node-name()

`node-name(node)` returns the node-name of the argument node.

## normalize-space()

`normalize-space(string)`, `normalize-space()` removes leading and trailing spaces from the specified string, and replaces all internal sequences of white space with one and returns the result. If there is no string argument it does the same on the current node.

For example: `normalize-space(' The XML ')` results in 'The XML'

## not()

`not(arg)` inverts its argument (converts true to false and vice versa). If the argument is not a Boolean, it is cast to one before its value is inverted.

## number()

`number(arg)` returns the numeric value of the argument. The argument could be a boolean, string, or node-set

For example: `number('100')` results in 100

It casts its argument to a number, using the following rules:

## one-or-more()

`one-or-more(item,item,...)` returns the argument if it contains one or more items, otherwise it raises an error.

## position()

`position()` returns the index position of the node that is currently being processed.

For example: `/book[position()<=3]` selects the first three book elements

## remove()

`remove((item,item,...),position)` returns a new sequence constructed from the value of the item arguments - with the item specified by the position argument removed.

For example: `remove(("ab", "cd", "ef"), 0)` results in ("ab", "cd", "ef")

For example: `remove(("ab", "cd", "ef"), 1)` results in ("cd", "ef")

For example: `remove(("ab", "cd", "ef"), 4)` results in ("ab", "cd", "ef")

## replace()

`replace(string,pattern,replace)` returns a string that is created by replacing the given pattern with the replace argument.

For example: `replace("Bella Italia", "l", "")` results in 'Be\*\*a Ita\*ia'

For example: `replace("Bella Italia", "l", "a")` results in 'Beaa Itaia'

## reverse()

`reverse((item,item,...))` returns the reversed order of the items specified.

For example: `reverse(("ab", "cd", "ef"))` results in ("ef", "cd", "ab")

For example: `reverse(("ab"))` results in ("ab")

## root()

`root(), root(node)` returns the root of the tree to which the current node or the specified belongs. This will usually be a document node.

## round()

`round(num)` rounds the number argument to the nearest integer.

For example: `round(3.14)` results in 3

## round-half-to-even()

`round-half-to-even()` Rounds the number argument to the nearest even number.

For example: `round-half-to-even(0.5)` results in 0

For example: `round-half-to-even(1.5)` results in 2

For example: `round-half-to-even(2.5)` results in 2

## **seconds-from-dateTime()**

`seconds-from-dateTime(datetime)` returns a decimal that represents the seconds component in the localized value of the argument.

For example: `seconds-from-dateTime(xs:dateTime("2005-01-10T12:30:00-04:10"))` results in 0

## **seconds-from-duration()**

`seconds-from-duration(datetimedur)` returns a decimal that represents the seconds component in the canonical lexical representation of the value of the argument.

## **seconds-from-time()**

`seconds-from-time(time)` returns an integer that represents the seconds component in the localized value of the argument.

For example: `seconds-from-time(xs:time("10:22:00"))` results in 0

## **starts-with()**

`starts-with(string1,string2)` returns true if string1 starts with string2, otherwise it returns false.

For example: `starts-with('XML','X')` results in true

## **string()**

`string(arg)` Converts its argument to a string. If a node set is passed as an argument, then the text value of the first node is returned.

For example: `string(314)` results in "314"

## **string-join()**

`string-join((string,string,...),sep)` returns a string created by concatenating the string arguments and using the sep argument as the separator.

For example: `string-join(('We', 'are', 'having', 'fun!'), ' ')` results in We are having fun!

For example: `string-join(('We', 'are', 'having', 'fun!'))` results in 'Wearehavingfun!'

For example: `string-join((), 'sep')` results in "

## **string-length()**

`string-length(string)`, `string-length()` returns the length of the specified string. If there is no string argument it returns the length of the string value of the current node.

For example: `string-length('Beatles')` results in 7

## **string-to-codepoints()**

`string-to-codepoints(string)` returns a sequence of code points from a string.

For example: `string-to-codepoints("Thérèse")` results in 84, 104, 233, 114, 232, 115, 101

## **subsequence()**

`subsequence((item,item,...),start,len)` returns a sequence of items from the position specified by the start argument and continuing for the number of items specified by the len argument. The first item is located at position 1.

For example: `subsequence(($item1, $item2, $item3,...), 3)` results in `($item3, ...)`

For example: `subsequence(($item1, $item2, $item3, ...), 2, 2)` results in `($item2, $item3)`

## **substring()**

`substring(string,start,len)`, `substring(string,start)` returns the substring from the start position to the specified length. Index of the first character is 1. If length is omitted it returns the substring from the start position to the end.

For example: `substring('Beatles',1,4)` results in 'Beat'

For example: `substring('Beatles',2)` results in 'eатles'

## **substring-after()**

`substring-after(string1,string2)` returns the remainder of string1 after string2 occurs in it.

For example: `substring-after('12/10','/')` results in '10'

## **substring-before()**

`substring-before(string1,string2)` returns the start of string1 before string2 occurs in it.

For example: `substring-before('12/10','/')` results in '12'

## **sum()**

`sum(arg,arg,...)` converts all nodes in a node set to a number and then adds up their value.

## **timezone-from-date()**

`timezone-from-date(date)` returns the time zone component of the argument if any.

## **timezone-from-datetime()**

`timezone-from-datetime(datetime)` returns the time zone component of the argument if any.

## **timezone-from-time()**

`timezone-from-time(time)` returns the time zone component of the argument if any.

## **tokenize()**

`tokenize(string,pattern)` is used to tokenize a String using a regular expression pattern.

For example, `tokenize("XPath is fun", "\s+")` results in ("XPath", "is", "fun")

You can also use an optional argument `flags`, which specifies one or more letters indicating options on how the matching is to be performed.

## trace()

`trace(value,label)` is used to debug queries.

## translate()

`translate(string1,string2,string3)` converts string1 by replacing the characters in string2 with the characters in string3.

For example: `translate('12:30','30','45')` results in '12:45'

For example: `translate('12:30','03','54')` results in '12:45'

For example: `translate('12:30','0123','abcd')` results in 'bc:da'

## true()

`true()` returns a true Boolean. It makes up for the lack of boolean literals in XPath.

For example: `true()` results in true

## upper-case()

Converts the string argument to upper-case.

## xpath()

Backbase custom function that evaluates an XPath expression given as a String parameter of a function.

## year-from-date()

`year-from-date(date)` returns an integer that represents the year in the localized value of the argument.

For example: `year-from-date(xs:date("2005-04-23"))` results in 2005

## year-from-datetime()

`year-from-datetime(datetimetz)` returns an integer that represents the year component in the localized value of the argument.

For example: `year-from-datetime(xs:dateTime("2005-01-10T12:30-04:10"))` results in 2005

## years-from-duration()

`years-from-duration(datetimedur)` returns an integer that represents the years component in the canonical lexical representation of the value of the argument.

## zero-or-one()

`zero-or-one(item,item,...)` returns the argument if it contains zero or one items, otherwise it raises an error.

## XPath Node Tests

### node()

Selects all child nodes, regardless of whether they are elements, attributes or text.

For example: `h2/node()` returns all nodes (text, elements, comments) of the `h2` element

### `text()`

Selects the text node, which is the maximum continuous run of text between other node types.

## XPath Variables

### `$_currentPath`

Local variable available in the `construct` event. When a `construct` event takes places, the `$_currentPath` contains the path to the loaded (or included) file.

### `$_dragCurrent`

Local variable. Contains the current element that is being dragged to another location in the BXML tree.

Available in the events; `drag-drop`, `drag-start`, and `drag-receive`. For an example, see `$bpc_dragSymbol`.

### `$_dragParent`

Local variable. Contains the parent element of an element that is being dragged to another location in the BXML tree.

Available in the events; `drag-drop`, `drag-start`, and `drag-receive`.

### `$_dragTarget`

Local variable. Contains the element which receives the element that has been dragged to another location in the BXML tree.

Available in the events; `drag-drop`, `drag-start`, and `drag-receive`.

### `$_element`

Local variable. Contains the element which is being moved to another location in the BXML tree.

Available in the `child-lost` and `receive` events: When an element is dragged or moved to another location in the BXML tree, a `child-lost` event is triggered on the parent element and a `receive` event on the receiving element.

### `$_eventSource`

Local variable. Contains the BXML element from which the event was triggered.

### `$_httpObject`

Local variable. Contains the HTTP object, mostly for use in the `s:script` tag to read HTTP headers or statuses.

Available after the `load` command is fired.

### `$_mouseElement`

Local variable. Contains the HTML element currently under the mouse when the event was fired.

## **\$\_mouseSource**

Local variable. Contains the BXML element currently under the mouse when the event was fired.

## **\$\_selectedText**

Global variable. Contains the current selected text in the browser window.

## **\$\_source**

Local variable. Contains the parent element of a child element which is being moved to another location in the BXML tree.

Available in the `child-lost` and `receive` events: When an element is dragged or moved to another location in the BXML tree, a `child-lost` event is triggered on the parent element and a `receive` event on the receiving element.

## **\$\_sourceIndex**

Local variable. Contains the child element index (an integer value) of the element which is being moved to another location in the BXML tree, that is whether it is the first, seconds, third etc. child of the parent.

Available in the `child-lost` and `receive` events: When an element is dragged or moved to another location in the BXML tree, a `child-lost` event is triggered on the parent element and a `receive` event on the receiving element.

## **\$\_target**

Local variable. Contains the element which receives the element that has been moved to another location in the BXML tree.

Available in the `child-lost` and `receive` events: When an element is dragged or moved to another location in the BXML tree, a `child-lost` event is triggered on the parent element and a `receive` event on the receiving element.

## **\$bpc\_bookmark**

Global variable. Contains the value of the requested bookmark reflecting the hash in your browser URL. It is the same as `document.location.hash` without the '#'. It is set on page startup, and is never changed when the hash is changed so it also reflects the bookmark (read-only).

## Implementation

```

<s:event b:on="construct" b:action="select" b:target="id($bpc_bookmark)" />
<s:behavior b:name="history" b:behavior="b-tab">
    <s:event b:on="select">
        <s:super />
        <s:history b:name="browser" b:bookmark="@id" />
            <s:task b:action="select" />
        </s:history>
    </s:event>
</s:behavior>
<b:tabbox style="height: 100%">
    <b:tab b:label="2001" id="2001" b:url="data/2001_short.xml" b:behavior="history" />
    <b:tab b:label="Seven Samurai" id="SevenSamurai" b:url="data/seven_short.xml" b:behavior="history" />
    <b:tab b:label="Life is Beautiful" id="LifeisBeautiful" b:state="selected" />

```

```
b:url="data/bella_short.xml" b:behavior="history" />
  <b:tab b:label="The Godfather" id="Godfather" b:url="data/godfather_short.xml"
b:behavior="history" />
</b:tabbox>
```

## \$bpc\_browser

Global variable. Contains the client browser being used (read-only). Possible values are: ie5, ie55, ie6, ie7, and moz.

## \$bpc\_controlpath

Global variable. Retrieves the path to the controls being used (read/write). By default, it is set to the \$bpc\_path + 'controls/dynamic'.

## \$bpc\_dragMargin

Indicates the margin that is needed to move your mouse (in pixels) before dragging is started (read/write). By default, it is set to 2.

## \$bpc\_dragOutline

Indicates the style of the dragoutline div (read/write). By default, it is set to border:1px dashed #000.

### Implementation

```
<s:event b:on="construct">
  <s:task b:action="assign" b:target="$bpc_dragOutline" b:select="'border: 2px red solid;'" />
</s:event>
<s:behavior b:name="dragreceive">
  <s:initatt b:dragreceive="dragobject" b:folder="true" b:open="true" />
</s:behavior>
<s:behavior b:name="drag">
  <s:initatt b:drag="dragobject" b:dragmode="outline" b:leaf="true" />
</s:behavior>
<b:tree b:behavior="dragreceive" b:label="Tree Control">
  <b:tree b:behavior="dragreceive" b:label="Friends">
    <b:tree b:behavior="drag" b:label="Clare" />
    <b:tree b:behavior="drag" b:label="Tom" />
    <b:tree b:behavior="drag" b:label="Alex" />
  </b:tree>
  <b:tree b:behavior="dragreceive" b:label="Colleagues">
    <b:tree b:behavior="drag" b:label="Sarah" />
    <b:tree b:behavior="drag" b:label="Mark" />
    <b:tree b:behavior="drag" b:label="Rachel" />
  </b:tree>
</b:tree>
```

## \$bpc\_dragSymbol

Global variable. Contains the BXML that is shown for the attribute b:dragmode="symbol"

### Implementation

In the following example, a b:tree control is defined in which you can drag the leaf elements and drop them into any folder element. Each leaf element uses the behavior called drag that sets its initial attributes so that each element is draggable, the drag mode is symbol, and that the element is a leaf. The behavior also has a drag-start event handler which is triggered when a draggable element is being dragged and defines the symbol that is displayed while the element is being dragged. The behavior dragreceive is used by the folder element to set initial attributes; that it can receive drag elements, it is a

folder element, and that all the folder are expanded by default. It also has a `dragreceive` event handler that defines the action that is taken when a draggable object is dropped: the current drag element is stored in the `$_dragCurrent` variable, and this is moved to the context node as its first child.

```
<s:behavior b:name="dragreceive">
    <s:initatt b:dragreceive="dragobject" b:folder="true" b:open="true" />
    <s:event b:on="drag-receive">
        <s:task b:action="move" b:destination=".." b:source="$_dragCurrent"
b:mode="asfirstchild" />
    </s:event>
</s:behavior>
<s:behavior b:name="drag">
    <s:initatt b:drag="dragobject" b:dragmode="symbol" b:leaf="true" />
    <s:event b:on="drag-start">
        <s:render b:destination="$bpc_dragSymbol" b:mode="replacechildren">
            
        </s:render>
    </s:event>
</s:behavior>
<b:tree b:behavior="dragreceive" b:label="Tree Control">
    <b:tree b:behavior="drag" b:label="John" />
    <b:tree b:behavior="dragreceive" b:label="Friends">
        <b:tree b:behavior="drag" b:label="Clare" />
        <b:tree b:behavior="drag" b:label="Tom" />
        <b:tree b:behavior="drag" b:label="Alex" />
        <b:tree b:behavior="drag" b:label="Mike" />
    </b:tree>
    <b:tree b:behavior="dragreceive" b:label="Colleagues">
        <b:tree b:behavior="drag" b:label="Sarah" />
        <b:tree b:behavior="drag" b:label="Mark" />
        <b:tree b:behavior="drag" b:label="Rachel" />
        <b:tree b:behavior="drag" b:label="Jane" />
    </b:tree>
</b:tree>
```

## \$bpc\_focusCurrent

Global variable. Contains the focusitem that currently has focus in the BXML space (is null when no element has focus).

## \$bpc\_focusCurrentElement

Global variable. Contains the focusitem that currently has focus in the HTML space (is null when no element has focus).

## \$bpc\_focusLastElement

Global variable. Contains the focusitem that previously had focus in the HTML space (is null when no element has focus).

## \$bpc\_fxTime

Global variable. Contains the duration of the animation in milliseconds of an `s:fxstyle` effect (read/write). The default is `500`.

## Implementation

```
<div style="position: absolute; border:1px solid red; width:50px; height:30px">Move me
    <s:event b:on="command">
        <s:fxstyle b:motion="linear" b:right="~-200px" />
        <s:fxstyle b:motion="linear" b:bottom="~-200px" />
    </s:event>
</div>
<br />
<br />
```

```
<br />
<br />
<div>
    <a b:action="assign" b:target="$bpc_fxTime" b:select="50">Click to speed up animation</a>
    <a b:action="assign" b:target="$bpc_fxTime" b:select="5000">Click to slow down
animation</a>
    <a b:action="alert" b:value="{$bpc_fxTime}">Get Animation Time</a>
</div>
```

## \$bpc\_fxTimeout

Global variable. Contains the animation interval in milliseconds of an `s:fxstyle` effect (read/write). The default is `10`. You can change this value, although specifying a low number consumes CPU.

### Implementation

```
<div style="position: absolute; border:1px solid red; width:50px; height:30px">Move me
    <s:event b:on="command">
        <s:fxstyle b:motion="linear" b:right="~-200px" />
        <s:fxstyle b:motion="linear" b:bottom="~-200px" />
    </s:event>
</div>
<br />
<br />
<br />
<br />
<div>
    <a b:action="assign" b:target="$bpc_fxTimeout" b:select="1">Click to speed up
interval</a>
    <a b:action="assign" b:target="$bpc_fxTimeout" b:select="5000">Click to slow down
interval</a>
    <a b:action="alert" b:value="{$bpc_fxTimout}">Get Interval Time</a>
</div>
```

## \$bpc\_hash

Global variable. Reflects the hash in your browser URL. It is the same as `document.location.hash` without the '#' (read-only).

## \$bpc\_keyAlt

Global variable. Boolean that is set to `true` if the Alt key is pressed.

## \$bpc\_keyCtrl

Global variable. Boolean that is set to `true` if the Ctrl key is pressed.

## \$bpc\_keyShift

Global variable. Boolean that is set to `true` if the Shift key is pressed.

## \$bpc\_mouseX

Global variable. Contains the X coordinate of the mouse (horizontal axis).

### Implementation

```
<div style="background-color: red; width: 500px; height: 500px; color: white">      Click
anywhere to view mouse coordinates
    <s:event b:on="command" b:action="alert" b:value="concat('X coordinate is ',
$bpc_mouseX, ', Y coordinate is ', $bpc_mouseY)" />
</div>
```

## \$bpc\_mouseY

Global variable. Contains the Y coordinate of the mouse (vertical axis).

### Implementation

```
<div style="background-color: red; width: 500px; height: 500px; color: white"> Click
anywhere to view mouse coordinates
    <s:event b:on="command" b:action="alert" b:value="{concat('X coordinate is ',
$bpc_mouseX, ', Y coordinate is ', $bpc_mouseY)}" />
</div>
```

## \$bpc\_path

Global variable. Retrieves the location of the BPC folder (read-only).

## \$bpc\_resizeLine

Global variable. Contains the `b:resizemode="line"` style properties of a resizable element. The default is `border:4px solid #AAA;`.

### Implementation

```
<s:event b:on="construct" b:action="assign" b:target="$bpc_resizeLine" b:select="'border:
10px red solid;'" />
<style type="text/css"> .movietable { border-width: 0px 0px 0px 1px;
border-style: solid; } .movietable th { border-width: 1px 1px 1px 0px;
border-style: solid; } .movietable thead tr { background-color: #eaeefb; }
.mvietable td { border-width: 0px 1px 1px 0px; border-style: solid;
} </style>
<table class="movietable" cellpadding="5" cellspacing="0" border="0">
<thead>
<tr>
<th b:resize="e" style="width:140px;">Film</th>
<th b:resize="e" style="width:140px;">Director</th>
<th b:resize="e" style="width:90px;">Genre</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pulp Fiction</td>
<td>Quentin Tarantino</td>
<td>Crime</td>
</tr>
</tbody>
</table>
```

## \$bpc\_scrollLeft

Global variable. Contains the distance in pixels of the vertical window scrollbar.

## \$bpc\_scrollTop

Global variable. Contains the distance in pixels of the horizontal window scrollbar.

## \$bpc\_tooltipClass

Indicates the class that is set for the `b:tooltiptext` attribute `tooltips` (read/write). By default, the value is `ToolTip`.

### Implementation

```
<style type="text/css"> .myClass { border: solid 3px red; } </style>
<a b:tooltiptext="A Tooltip">Hover over the element...click to change the class
    <s:event b:on="command">
        <s:task b:action="assign" b:target="$bpc_tooltipClass" b:select="'myClass'" />
    </s:event>
```

```
</a>
```

## \$bpc\_tooltipTimeout

Defines how long a tooltip is displayed (read/write). By default, it is set to 25 milliseconds.

### Implementation

```
<a b:tooltiptext="The should appear in 2 seconds">Hover over the element...
  <s:event b:on="construct">
    <s:task b:action="assign" b:target="$bpc_tooltipTimeout" b:select="2000" />
  </s:event>
</a>
```

## \$bpc\_version

Retrieves the version number of the BPC engine (read-only).

### Implementation

```
<div>See BPC Version
  <s:event b:on="command">
    <s:task b:action="alert" b:value="${bpc_version}" />
  </s:event>
</div>
```

## \$bpc\_windowHeight

Global variable. Contains the window height dimension in pixels.

## \$bpc\_windowWidth

Global variable. Contains the window width dimension in pixels.

## \$bpc\_xsltParser

Global variable. Boolean variable used when performing browser templating that tests the XML version (XSLT engine) supported by the browser: `true` indicates the browser supports XSLT 1.0, `false` indicates XSL.

### Implementation

In the following example, the behavior defines a procedural loop that first tests the supported browser XSLT engine, and depending on the result loads either the `table.xslt` stylesheet or `table-ie5.xslt`.

```
<s:execute>
  <s:task b:action="show" />
</s:execute>
<div b:behavior="browsertemplating"></div>
<s:behavior b:name="browsertemplating" b:focusgroup="true">
  <s:event b:on="construct">
    <s:variable b:name="data" b:scope="local" />
    <s:variable b:name="xslt" b:scope="local" />
    <s:choose>
      <s:when b:test="$bpc_xsltParser">
        <s:task b:action="load" b:destination="$xslt" b:url="table.xslt" />
      </s:when>
      <s:otherwise>
        <s:task b:action="load" b:destination="$xslt" b:url="table-ie5.xslt" />
      </s:otherwise>
    </s:choose>
    <s:task b:action="load" b:destination="$data" b:url="films.xml" />
    <s:task b:action="string2xml" b:variable="$xslt" />
```

```
<s:task b:action="string2xml" b:variable="$data" />
<s:task b:action="xsl-transform" b:stylesheet="$xslt" b:datasource="$data"
b:destination="." />
</s:event>
</s:behavior>
```