potix

PRELIMINARY

SIMPLY RICH

ZK™

The Developer's Reference

Version 2.1.0

July 2006

Potix Corporation

Revision 16

Table of Contents

1. Introduction

Welcome to ZK, the simplest way to make Web applications rich.

The Developer's Reference fully describes properties and methods of components. For concepts, features, refer to **the Developer's Guide**. For installation, refer to **the Quick Start Guide**.

2. The ZK User Interface Markup Language

Implicit Objects

For scripts (aka., zsccript) and EL expressions embedded in a ZUML page, there are a set of implicit objects that enable developers to access components more efficiently.

applicationScope - `java.util.Map`

A map of custom attributes associated with the Web application. It is the same as the `getAttributes` method in the `com.potix.zk.ui.WebApp` interface.

A Web application is a WAR, and each Web application has an independent set of custom attributes. These attributes are used mainly to communicate among different desktops and sessions.

If the client is based on HTTP, such as a Web browser, this is the same map of attributes stored in `javax.servlet.ServletContext`. In other words, you could use it communicate with other servlets, such as JSF.

arg - `java.util.Map`

The `arg` argument passed to the `createComponents` method in the `com.potix.zk.ui.Executions` class. It might be `null`, depending on how `createComponents` is called.

It is the same as `self.desktop.execution.arg`.

```
params.put("name", "John");
Executions.createComponents("/my.zul", null, params);
```

Then, in my.zul,

```
<window title="${arg.name}">
...
```

Notice that `arg` is available only when creating the components for the included page, say `my.zul`. On the other hand, all events, including `onCreate`, are processed later. Thus, if you want to access `arg` in the `onCreate`'s listener, use the `getArg` method of the `com.potix.zk.ui.event.CreateEvent` class.

componentScope - `java.util.Map`

A map of custom attributes associated with the component. It is the same as the `getAttributes` method in the `com.potix.zk.ui.Component` interface.

desktop - `com.potix.zk.ui.Desktop`

The current desktop. It is the same as `self.desktop`.

```
desktop.getPage("main");
```

desktopScope - `java.util.Map`

A map of custom attributes associated with the desktop. It is the same as the `getAttributes` method in the `com.potix.zk.ui.Desktop` interface.

It is mainly used to communicate among pages in the same desktop.

each - `java.lang.Object`

The current item of the collection being iterated, when ZK evaluates an iterative element. An iterative element is an element with the `forEach` attribute.

```
<listbox width="100px">
  <listitem label="${each}" forEach="${contacts}"/>
</listbox>
```

event - `com.potix.zk.ui.event.Event` or derived

The current event. Available for the event listener only.

```
<textbox onChanging="react(event.value)"/>
<combobox onChanging="autoComplete()"/>
<zscript>
void react(String value) {
...
}
void autoComplete() {
  String value = event.getValue();
...
}
</zscript>
```

forEachStatus – `com.potix.zk.ui.util.ForEachStatus`

The status of an iteration. ZK exposes the information relative to the iteration taking place when evaluating the iterative element.

```
<zk>
  <zscript>
grades = new String[] {"Best", "Better", "Good"};
  </zscript>
  <listbox width="100px">
    <listitem label="${forEachStatus.index}: ${each}" forEach="${grades}"/>
  </listbox>
```

```
</zk>
```

page - com.potix.zk.ui.Page

The current page. It is the same as `self.page`.

pageScope - java.util.Map

A map of custom attributes associated with the current page. It is the same as the `getAttributes` method in the `com.potix.zk.ui.Page` interface.

self - com.potix.zk.ui.Component

The component itself. In other words, it is the closest component, depicted as follows.

```
<listbox>
  <zscript>self.getItems();</zscript><!-- self is listbox -->
  <listitem value="ab" label="${self.value}"/><!-- self is listitem -->
  <zscript>self.getSelectedIndex();</zscript><!-- self is listbox -->
</listbox>
```

session - com.potix.zk.ui.Session

The session. It is similar to `javax.servlet.http.HttpSession`[1].

sessionScope - java.util.Map

A map of custom attributes associated with the session. It is the same as the `getAttributes` method in the `com.potix.zk.ui.Session` interface.

If the client is based on HTTP, such as a Web browser, this is the same map of attributes stored in `javax.servlet.http.HttpSession`. In other words, you could use it communicate with other servlets, such as JSF.

spaceOwner - com.potix.zk.ui.IdSpace

The space owner of this component. It is the same as `self.spaceOwner`.

spaceScope - java.util.Map

A map of custom attributes associated with the ID space containing this component.

1 ZK session actually encapsulates the HTTP session to make ZK applications independent of HTTP.

Processing Instructions

The XML processing instructions describe how to process the ZUML page. They will be processed first before processing XML elements.

The `component` Directive

```
<?component name="myName" macro-uri="/mypath/my.zul"
  [prop1="value1"] [prop2="value2"]... ?>

<?component name="myName" [class="myPackage.myClass"]
  [extend="true"] [mold-name="myMoldName"] [mold-uri="/myMoldUri"]
  [prop1="value1"] [prop2="value2"]... ?>
```

Defines a new component. There are two formats: by-macro and by-class.

The by-macro Format

```
<?component name="myName" macro-uri="/mypath/my.zul"
  [prop1="value1"] [prop2="value2"]... ?>
```

You could define a new component based on a ZUML page. It is also called the *macro component*. In other words, once an instance of the new component is created, it creates child components based on the specified ZUML page (the `macro-uri` attribute).

In additions, you could specify the initial properties (such as `prop1` in the above example), such that they are always passed to the macro component (thru the `arg` variable).

The by-class Format

```
<?component name="myName" [class="myPackage.myClass"]
  [extend="true"] [mold-name="myMoldName"] [mold-uri="/myMoldUri"]
  [prop1="value1"] [prop2="value2"]...?>
```

In addition to defining a component by a ZUML page (aka., a macro component), You could define a new component by implementing a class that implements the `com.potix.zk.ui.Component` interface. Then, use the `by-class` format to declare such kind of components for a page.

To define a new component, you have to specify at least the `class` attribute, which is used by ZK to instantiate a new instance of the component.

In addition to defining a new component, you can override properties of existent components by specifying `extend="true"`. In other words, if `extend="true"` is specified, the previous definition of the component (with the same name) is loaded as the default value and then override only properties that are specified in this directive.

For example, assume you want to use `MyWindow` instead of the default window,

com.potix.zul.html.Window, for all windows defined in this ZUML page. Then, you can declare it as follows.

```
<?component name="window" extend="true" class="MyWindow"?>
...
<window>
...
</window>
```

It is equivalent to the following codes.

```
<window use="MyWindow">
...
</window>
```

In addition, you could specify the properties to initialize. For example, you want to use the style class called blue for all buttons used in this page, then you could:

```
<?component name="button" extend="true" sclass="blue"?>
```

Similarly, you could use the following definition to use OK as the default label for all buttons specified in this page.

```
<?component name="button" extend="true" label="OK"?>
```

Notice that the properties won't be applied if a component is created manually (by zscript or by Java codes). If you still want them to be applied with the initialial properties, you could invoke the applyProperties method as follows.

```
<zscript>
   Button btn = new Button();
   btn.applyProperties(); //apply the initial properties
</zscript>
```

class

[Optional]

Used to specify the class to instantiate an instance of such kind of components. Unlike other directives, the class can be defined with zscript.

extend

[Optional]

If specified with "true", the existent definition will be loaded to initialize the new component definition. In other words, it *extends* the existent definition instead of defining a brand-new one.

macro-uri

[Required if the by-macro format is used]

Used with the by-macro format to specify the URI of the ZUML page, which is used as the template to create components.

mold-name

[Optional][Default: default]

Used with the by-class format to specify the mold name. If `mold-name` is specified, `mold-uri` must be specified, too.

mold-uri

[Optional]

Used with the by-class format to specify the mold URI. If `mold-uri` is specified but `mold-name` is not specified, the mold name is assumed as `default`.

name

[Required]

The component name. If an existent component is defined with the same name, the existent component is completely invisible in this page. If the by-class format is used, the attributes of the existent components are used to initialize the new components and then override with what are defined in this processing instruction.

The `init` Directive

```
<?init class="..." [arg0="..."] [arg1="..."] [arg2="..."] [arg3="..."]?>
```

```
<?init zscript="..." [arg0="..."] [arg1="..."] [arg2="..."] [arg3="..."]?>
```

There are two formats. The first format is to specify a class that is used to do the application-specific initialization. The second format is to specify a `zscript` file to do the application-specific initialization.

The initialization takes place before the page is evaluated and attached to a desktop. Thus, the `getDesktop`, `getId` and `getTitle` method will return null, when initializing. To retrieve the current desktop, you could use the `com.potix.zk.ui.Execution` interface.

You could specify any number of the `init` directive. The specified class must implement the `com.potix.zk.ui.util.Initator` interface.

```
<?init class="MyInit1"?>
<?init class="MyInit2"?>
```

class

[Optional]

A class name that must implement the `com.potix.zk.ui.util.Initator` interface. Unlike the `init` directive, the class name cannot be the class that is defined in zscript codes.

An instance of the specified class is constructed and its `doInit` method is called in the Page Initial phase (i.e., before the page is evaluated). The `doFinally` method is called after the page has been evaluated. The `doCatch` method is called if an exception occurs during the evaluation.

Thus, you could also use it for cleanup and error handling.

zscript

[Optional]

A `script` file that will be evaluated in the Page Initial phase.

arg0, arg1...

[Optional]

You could specify any number of arguments. It will be passed to the `doInit` method if the first format is used, or as the `args` variable if the second format is used. Note: the first argument is `arg0`, the second is `arg1` and follows.

The `page` Directive

```
<?page [id="..."] [title="..."] [style="..."] [language="xul/html"]?>
```

It describes attributes of a page.

id

[Optional][Default: *generated automatically*]

Specifies the identifier of the page, such that we can retrieve it back. If an alphabetical identifier is assigned, it will be available to scripts (aka., `zscript`) and EL expressions embedded in ZUML pages.

```
<?page id="${param.id}"?>
```

title

[Optional][Default: *none*]

Specifies the page title that will be shown as the title of the browser.

It can be changed dynamically by calling the `setTitle` method in the `com.potix.zk.ui.Page` interface.

```
<?page title="${param.title}"?>
```

style

[Optional][Default: `width:100%`]

Specifies the CSS style used to render the page. If not specified, it depends on the mold. The default mold uses `width:100%` as the default value.

```
<?page style="width:100%;height:100%"?>
```

language

[Optional][Default: *depending on the extension*][`xul/html` | `xhtml`]

Specifies the language of this page.

Currently, it supports `xul/html` and `xhtml`.

The `taglib` Directive

```
<?tablib uri="/myURI" prefix="my"?>
```

This directive is used to load a `taglib` file, which defines a set of EL functions. The format of a `taglib` file is the same as that of JSP `taglib` files.

In the following example, we loads functions defined in `core.dsp.tld` and then use the function called `l`.

```
<?taglib uri="/WEB-INF/tld/web/core.dsp.tld" prefix="c"?>
<window title="${c:l('my.title')}">
...
</window>
```

uri

[Required]

A URL of the `taglib` file. Unlike other URL and URI, it doesn't interpret ~ or * specially. And, the page and the `taglib` files it references must be in the same Web application.

prefix

[Required]

A prefix used to identify functions defined in this `taglib` file. The prefix could be any non-empty string.

The `variable-resolver` Directive

```
<?variable-resolver class="..."?>
```

Specifies the variable resolver that will be used by the `zscript` interpreter to resolve unknown variables. The specified class must implement the `com.potix.zk.ui.util.VariableResolver` interface.

You can specify multiple variable resolvers with multiple `variable-resolver` directives. The later declared one has higher priority.

Notice that the `variable-resolver` directives are evaluated before the `init` directives, so the `zscript` codes referenced by the `init` directives are affected by the variable resolver.

The following is an example when using ZK with the Spring framework. It resolves Java Beans declared in the Spring framework, such that you access them directly.

```
<?variable-resolver class="com.potix.zkplus.spring.DelegatingVariableResolver"?>
```

class

[Optional]

A class name that must implement the `com.potix.zk.ui.util.VariableResolver` interface. Unlike the `init` directive, the class name cannot be the class that is defined in zscript codes.

The `xml-stylesheet` Directive

`<?xml-stylesheet href="`*`/myURI`*`" [type="text/css"]?>`

This directive is used to specify the style sheet to be loaded with this page. You could specify any number of style sheets by use of this directive.

href

[Required]

A URL of the style sheet. Like other URL and URI, it has several characteristics as follows.

1. It is relative to the servlet context path (aka., the `getContextPath` method from the `javax.servlet.http.HttpServletRequest` interface). In other words, ZK will prefix it with the servlet context automatically.
2. It resolves "~" to other Web application (aka., different ServletContext). Notice that Web server administrator might disable Web applications from peeking other's content[2].
3. It accepts "*" for loading browser and Locale dependent style sheet.

The algorithm to resolve "*" is as follows.

1. If there is one "*" is specified in an URL or URI such as `/my*.css`, then "*" will be

2 Refer to the `getContext` meth from the `javax.servlet.ServletContext` interface.

replaced with a proper Locale depending on the preferences of user's browser.
For example, user's preferences is `de_DE`, then ZK searches `/my_de_DE.css`, `/my_de.css`, and `/my.css` one-by-one from your Web site, until any of them is found. If none of them is found, `/my.css` is still used.

1. If two or more "*" are specified in an URL or URI such as "/my*/lang*.css", then the first "*" will be replaced with "`ie`" for Internet Explorer and "`moz`" for other browsers[3].
   If the last "*" will be replaced with a proper Locale as described above.
2. All other "*" are ignored.

type

[Optional][Default: text/css]

Specifies the type of the style sheet. Currently, it supports only text/css.

ZK Elements

ZK elements are special XML elements that are used to control ZUML pages other than creating components.

The XML Namespace

If there is name conflicts, you could specify the XML name space:

`http://www.potix.com/2005/zk`

```
<zk:attribute xmlns:zk="http://www.potix.com/2005/zk">
...
```

The `attribute` Element

`<attribute name="`*myName*`">`*myValue*`</attribute>`

It defines a XML attribute of the enclosing element. The content of the element is the attribute value, while the `name` attribute specifies the attribute name. It is useful if the value of an attribute is sophisticated, or the attribute is conditional.

```
<button label="Hi">
  <attribute name="onClick">alert("Hi")</attribute>
</button>
```

It is equivalent to

```
<button label="Hi" onClick="alert(&amp;quot;Hi&amp;quot;)"/>
```

3 In the future editions, we will use different codes for browsers other than IE and FF.

Another example:

```
<button>
  <attribute name="label" if="${param.happy}">Hello World!</attribute>
</button>
```

name

[Required]

Specifies the attribute name.

if

[Optional][Default: `true`]

Specifies the condition to evaluate this element. This element is ignored if the value specified to this attribute is evaluated to false.

unless

[Optional][Default: `false`]

Specifies the condition *not* to evaluate this element. This element is ignored if the value specified to this attribute is evaluated to true.

The `custom-attributes` Element

```
<custom-attributes
    [scope="component|space|page|desktop|session|application]
    attr1="value1" [attr2="value2"...]/>
```

It defines a set of custom attributes of the specified scope. You could specify as many as attributes you want. These attributes can be retrieve by the `getAttribute` method of the `Component` interface with the specified scope.

```
<custom-attributes cd="${param.cd}" a.b="ab"/>
```

scope

[optional][Default: `component`]

Specifies the scope to which the custom attributes are associated. If not specified, the component enclosing this element is the default scope to use.

if

[Optional][Default: `true`]

Specifies the condition to evaluate this element. This element is ignored if the value specified to this attribute is evaluated to false.

unless

[Optional][Default: `false`]

Specifies the condition *not* to evaluate this element. This element is ignored if the value specified to this attribute is evaluated to true.

The `zk` Element

`<zk>...</zk>`

It is a special element used to aggregate other components. Unlike a real component (say, `hbox` or `div`), it is not part of the component tree being created. In other words, it doesn't represent any component. For example,

```
<window>
   <zk>
      <textbox/>
      <textbox/>
   </zk>
</window>
```

is equivalent to

```
<window>
   <textbox/>
   <textbox/>
</window>
```

The main use is to represent multiple root elements in XML format.

```
<?page title="Multiple Root"?>
<zk>
   <window title="First">
   ...
   </window>
   <window title="Second" if="${param.secondRequired}">
   ...
   </window>
</zk>
```

The other use is to iterate over versatile components.

```
<window>
   <zk forEach="${mycols}">
      <textbox if="${each.useText}"/>
      <datebox if="${each.useDate}"/>
      <combobox if="${each.useCombo}"/>
   </zk>
</window>
```

if

[Optional][Default: `true`]

Specifies the condition to evaluate this element. This element is ignored if the value specified to this attribute is evaluated to false.

unless

[Optional][Default: `false`]

Specifies the condition *not* to evaluate this element. This element is ignored if the value specified to this attribute is evaluated to true.

forEach

[Optional][Default: *ignored*]

It specifies a collection of objects, such that the `zk` element will be evaluated repeatedly against each object in the collection. If not specified or empty, this attribute is ignored. If non-collection object is specified, it is evaluated only once as if a single-element collection is specified.

The `zscript` Element

```
<zscript>Java codes</zscript>
<zscript src="uri"/>
```

It defines a piece of Java codes that will be interpreted when the page is evaluated. It has two formats as shown above. The first format is used to embed Java codes directly in the page. The second format is used to reference an external file that contains Java codes.

```
<zscript>
alert("Hi");
</zscript>
<zscript src="/codes/my.bs"/>
```

Like other ZK elements, it is not a component but a special XML element.

src

[Optional][Default: *none*]

Specifies the URI of the file containing Java codes. If specified, the Java codes will be loaded as if they are embedded directly.

Note: the file shall contain the Java source codes that can be interpreted by BeanShell. Don't specify a class file (aka. byte codes).

Like other URI, it accepts "*" and prefixes servlet context path automatically. Refer to the `href` attribute in the **`xml-stylesheet` Directive** section for details.

if

[Optional][Default: `true`]

Specifies the condition to evaluate this element. This element is ignored if the value specified to this attribute is evaluated to false.

unless

[Optional][Default: `false`]

Specifies the condition *not* to evaluate this element. This element is ignored if the value specified to this attribute is evaluated to true.

ZK Attributes

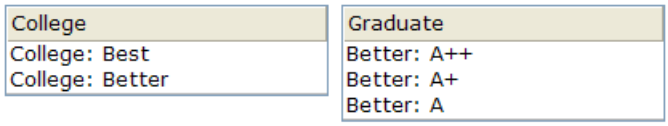
ZK attributes are used to control the associated element, other than initializing the data member.

The `forEach` Attribute

It specifies a collection of objects, such that the associated element will be evaluated repeatedly against each object in the collection. If not specified or empty, this attribute is ignored, and the element is evaluated only once. If non-collection object is specified, it is evaluated only once as if a single-element collection is specified.

For each iteration, two variables, `each` and `forEachStatus`, are assigned automatically to let developers control how to evaluate the associated element.

```
<hbox>
  <zscript>
classes = new String[] {"College", "Graduate"};
grades = new Object[] {
  new String[] {"Best", "Better"}, new String[] {"A++", "A+", "A"}
};
  </zscript>
  <listbox width="200px" forEach="${classes}">
    <listhead>
      <listheader label="${each}"/>
    </listhead>
    <listitem label="${forEachStatus.previous.each}: ${each}"
    forEach="${grades[forEachStatus.index]}"/>
  </listbox>
</hbox>
```



The `if` Attribute

It specified the condition to evaluate the associated element. In other words, the associated element and all its child elements are ignored, if the condition is evaluated to false.

The `unless` Attribute

It specified the condition *not* to evaluate the associated element. In other words, the associated element and all its child elements are ignored, if the condition is evaluated to true.

The `use` Attribute

It specifies a class to create a component instead of the default one. In the following example, `MyWindow` is used instead of the default class, `com.potix.zul.html.Window`.

```
<window use="MyWindow"/>
```

3. EL Expressions

This chapter describes the details about applying EL expressions to ZUML pages.

Overview

EL expressions use the syntax `${expr}`. For example,

```
<element attr1=”${bean.property}”.../>
${map[entry]}
<another-element>${3+counter} is ${empty map}</another-element>
```

When an EL expression is used as an attribute value, it could return any kind of objects as long as the component accepts it. For example, the following expression will be evaluated to a Boolean object.

```
<window if="${some > 10}">
```

Using EL Expressions

EL expressions can be used

- In static text
- In any attribute's value including XML elements and XML processing instructions.

Variables

Implicit Objects

Literals

Operators

Functions

Using Functions

Defining Functions

Standard Implicit Objects that ZK supports

Like using EL expressions in JSP pages, you could use most of standard implicit objects in ZUML pages.

applicationScope - `java.util.Map`

A map of application-scoped attributes (String, Object).

cookie - `java.util.Map`

A map of cookies of the request. (String, Cookie).

header - `java.util.Map`

A map of headers of the request. (String, String).

headerValues - `java.util.Map`

A map of headers of the request. (String, String[]).

pageContext - `javax.servlet.jsp.PageContext`

The page context.

pageScope - `java.util.Map`

A map of page-scoped attributes.

Notice: the page concept is a bit different from JSP because a ZK page exists across requests.

param - `java.util.Map`

A map of parameters of the request. (String, String).

paramValues - `java.util.Map`

A map of parameters of the request. (String, String[]).

requestScope - `java.util.Map`

A map of request-scoped attributes (String, String).

sessionScope - `java.util.Map`

A map of session-scoped attributes (String, String).

ZK Implicit Objects

All variables defined in ZK scripts (aka., zscript) are available for the EL expressions. Thus, all implicit objects described in the previous chapter are also the implicit objects for the EL expressions. You are free to use `self`, `event`, `componentScope` and others. Refer to the **Implict Objects** section in the **ZK User Interface Markup Language** chapter.

4. The XUL Components

Overview

- All XUL components are packed in the `com.potix.zul.html` package.
- The XML name space is `http://www.potix.com/2005/zul`
- The extensions include `xul` and `zul`.
- The component names are case-sensitive. They are all in lower-cases.

XulElement

All XHTML components are derived from the `com.potix.zul.html.impl.XulElement` class.

Components

Audio

Box

Button

Caption

Checkbox

Column

Columns

Combobox

Comboitem

Datebox

Decimalbox

Div

Grid

Groupbox

Hbox

Html

Iframe

Image

Include

Intbox

Label

Listbox

Listcell

Listfoot

Listfooter

Listhead

Listheader

Listitem

Menu

Menubar

Menuitem

Menupopup

Menuseparator

Popup

Popupset

Radio

Radiogroup

Row

Rows

Separator

Slider

Space

Tab

Tabbox

Tabpanel

Tabpanels

Tabs

Textbox

Timer

Toolbar

Toobarbutton

Tree

Treecell

Treechildren

Treecol

Treecols

Treeitem

Treerow

Vbox

Window

Supplemental Classes

AbstractListModel

Constraint

Constrainted

Fileupload

ListModel

ListitemRenderer

Messagebox

RendererCtrl

SimpleContraint

SimpleListModel

5. The XHTML Components

Overview

- All XHTML components are packed in the `com.potix.zhtml` package.
- The XML name space is `http://www.w3.org/1999/xhtml`
- The extensions include `htm`, `html`, `xhtml` and `zhtml`.
- The component names are case-insensitive. Developers could use any combination of lower or upper cases.

URL and encodeURL

A XHTML component generates attributes directly to native HTML tags. It means, unlike XUL, it doesn't prefix the servlet context path to attributes for specifying URL. For example, the following codes don't work (unless the servlet context is "").

```
<img href="/my/good.png"/>
```

Rather, you shall use the `encodeURL` function in EL expressions as follows.

```
<?taglib uri="/WEB-INF/tld/web/core.dsp.tld" prefix="p"?>
...
<img href="${p:encodeURL('/my/good.png')}"/>
```

In Java, you shall use the `encodeURL` method from `com.potix.zk.ui.Execution`.

```
<img id="another"/>
<zscript>
  another.setDynamicAttribute("href",
    Executions.getCurrent().encodeURL("/my/good.png"));
</zscript>
```

Notice that XUL components and all ZK features that accept an URL will invoke the `encodeURL` method automatically[4].

AbstractTag

All XHTML components are derived from the `com.potix.zhtml.impl.AbstractTag` class.

A XHTML component is a thin wrapper that encapsulates a native HTML tag. It is different from a XUL component or other none-native component in several ways.

- By implementing the `com.potix.zk.ui.ext.RawId` interface, the universal identifier (`getUuid`) is the same as the identifier (getId).

4 The reason not to handle XHTML compoents is that we don't know which attribute requires URL.

- By implementing the `com.potix.zk.ui.ext.DynamicAttributes` interface, all XHTML components support arbitrary attributes. In other words, any attribute name is legal (as long as the targeted browser supports).

Raw

A special component, `com.potix.zhtml.Raw`, used to represent any component that is not declared in the following section (i.e., not in lang.xml). In other words, if any unrecognized component name is found, an instance of `Raw` is created, such that a proper HTML tag will be generated correspondingly. In other words, any component name is legal (as long as the targeted browser supports).

```
<marquee align="top">...</marquee>
```

It is equivalent to

```
new Raw().setDynamicAttribute("align", "top");
```

Components

A

Abbr

Acronym

Address

Area

B

Base

Big

Blockquote

Body

Br

Button

Caption

Cite

Code

Collection

Colgroup

Dd

Del

Dfn

Dir

Div

Dl

Dt

Em

Embed

Fieldset

Font

Form

H1

H2

H3

H4

Head

Hr

Html

I

Iframe

Img

Input

Ins

Isindex

Kbd

Label

Legend

Li

Link

Map

Menu

Meta

Nobr

Object

Ol

Optgroup

Option

P

Pre

Q

S

Sam

Script

Select

Small

Span

Strong

Style

Sub

Sup

Table

Tbody

Td

Text

Textarea

Tfoot

Th

Thead

Title

Tr

Tt

Ul

Var

Supplement Classes

Fileupload

Messagebox