potix

PRELIMINARY

SIMPLY RICH

ZK™

The Developer's Reference

Version 2.1.2

September 2006

Potix Corporation

Revision 27

Table of Contents

1. Introduction....10

2. The ZK User Interface Markup Language....11

Implicit Objects.... 11
applicationScope - java.util.Map.... 11
arg - java.util.Map.... 11
componentScope - java.util.Map.... 11
desktop - org.zkoss.zk.ui.Desktop.... 12
desktopScope - java.util.Map.... 12
each - java.lang.Object.... 12
event - org.zkoss.zk.ui.event.Event or derived.... 12
forEachStatus – org.zkoss.zk.ui.util.ForEachStatus.... 12
page - org.zkoss.zk.ui.Page.... 13
pageScope - java.util.Map.... 13
requestScope – java.util.Map.... 13
self - org.zkoss.zk.ui.Component.... 13
session - org.zkoss.zk.ui.Session.... 13
sessionScope - java.util.Map.... 13
spaceOwner - org.zkoss.zk.ui.IdSpace.... 14
spaceScope - java.util.Map.... 14

Processing Instructions.... 14
The component Directive.... 14
The import Directive.... 16
The init Directive.... 17
The link and meta Directives.... 18
The page Directive.... 19
The taglib Directive.... 19
The variable-resolver Directive.... 20

ZK Elements.... 21
The XML Namespace.... 21
The attribute Element.... 21
The custom-attributes Element.... 22
The zk Element.... 22
The zscript Element.... 24

ZK Attributes.... 26
The forEach Attribute.... 26
The forEachBegin Attribute.... 26

The forEachEnd Attribute..... 26
The if Attribute..... 27
The unless Attribute..... 27
The use Attribute..... 27

3. EL Expressions..... 28

Overview..... 28
Using EL Expressions..... 28
Variables..... 28
Implicit Objects..... 28
Literals..... 28
Operators..... 28
Functions..... 28

Standard Implicit Objects that ZK supports..... 29
applicationScope - java.util.Map..... 29
cookie - java.util.Map..... 29
header - java.util.Map..... 29
headerValues - java.util.Map..... 29
pageContext - javax.servlet.jsp.PageContext..... 29
pageScope - java.util.Map..... 29
param - java.util.Map..... 29
paramValues - java.util.Map..... 30
requestScope - java.util.Map..... 30
sessionScope - java.util.Map..... 30

ZK Implicit Objects..... 30

4. The XUL Components..... 31

Overview..... 31
XulElement..... 31

Components..... 31
Audio..... 31
Box..... 31
Button..... 31
Caption..... 31
Checkbox..... 31
Column..... 31
Columns..... 32
Combobox..... 32
Comboitem..... 32

Datebox..... 32
Decimalbox..... 32
Div..... 32
Grid..... 32
Groupbox..... 32
Hbox..... 32
Html..... 32
Iframe..... 32
Image..... 32
Include..... 33
Intbox..... 33
Label..... 33
Listbox..... 33
Listcell..... 33
Listfoot..... 33
Listfooter..... 33
Listhead..... 33
Listheader..... 33
Listitem..... 33
Menu..... 33
Menubar..... 33
Menuitem..... 34
Menupopup..... 34
Menuseparator..... 34
Popup..... 34
Popupset..... 34
Radio..... 34
Radiogroup..... 34
Row..... 34
Rows..... 34
Separator..... 34
Slider..... 34
Space..... 34
Splitter..... 35
Style..... 35
Tab..... 35
Tabbox..... 35
Tabpanel..... 35
Tabpanels..... 35
Tabs..... 35

Textbox..... 35
Timer..... 35
Toolbar..... 35
Toobarbutton..... 35
Tree..... 35
Treecell..... 36
Treechildren..... 36
Treecol..... 36
Treecols..... 36
Treeitem..... 36
Treerow..... 36
Vbox..... 36
Window..... 36
Supplemental Classes..... 36
AbstractListModel..... 36
Constraint..... 36
Constrainted..... 36
Fileupload..... 37
ListModel..... 37
ListitemRenderer..... 37
Messagebox..... 37
RendererCtrl..... 37
SimpleContraint..... 37
SimpleListModel..... 37
5. The XHTML Components..... 38
Overview..... 38
URL and encodeURL..... 38
AbstractTag..... 38
Raw..... 39
Components..... 39
A..... 39
Abbr..... 39
Acronym..... 39
Address..... 39
Area..... 39
B..... 39
Base..... 40
Big..... 40

Blockquote..... 40
Body..... 40
Br..... 40
Button..... 40
Caption..... 40
Cite..... 40
Code..... 40
Collection..... 40
Colgroup..... 40
Dd..... 40
Del..... 41
Dfn..... 41
Dir..... 41
Div..... 41
Dl..... 41
Dt..... 41
Em..... 41
Embed..... 41
Fieldset..... 41
Font..... 41
Form..... 41
H1..... 41
H2..... 42
H3..... 42
H4..... 42
Head..... 42
Hr..... 42
Html..... 42
I..... 42
Iframe..... 42
Img..... 42
Input..... 42
Ins..... 42
Isindex..... 42
Kbd..... 43
Label..... 43
Legend..... 43
Li..... 43
Link..... 43
Map..... 43

Menu..... 43
Meta..... 43
Nobr..... 43
Object..... 43
Ol..... 43
Optgroup..... 43
Option..... 44
P..... 44
Pre..... 44
Q..... 44
S..... 44
Sam..... 44
Script..... 44
Select..... 44
Small..... 44
Span..... 44
Strong..... 44
Style..... 44
Sub..... 45
Sup..... 45
Table..... 45
Tbody..... 45
Td..... 45
Text..... 45
Textarea..... 45
Tfoot..... 45
Th..... 45
Thead..... 45
Title..... 45
Tr..... 45
Tt..... 46
Ul..... 46
Var..... 46
Supplement Classes..... 46
Fileupload..... 46
Messagebox..... 46
Appendix A. WEB-INF/web.xml..... 47
ZK Loader..... 47
The Initial Parameters..... 47

ZK AU Engine..... 47
ZK Session Cleaner..... 47
ZK Filter..... 48
The Initial Parameters..... 48
How to Specify in web.xml..... 48
Sample of web.xml..... 49
Appendix B. WEB-INF/zk.xml..... 51
Overview..... 51
The listener Element..... 51
The log Element..... 55
The desktop-config Element..... 55
The session-config Element..... 56
The language-config Element..... 57
The system-config Element..... 57
The el-config Element..... 60
The error-page Element..... 60
The preference Element..... 60

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 `org.zkoss.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 `org.zkoss.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 `org.zkoss.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 `org.zkoss.zk.ui.Component` interface.

desktop - `org.zkoss.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 `org.zkoss.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 - `org.zkoss.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 – `org.zkoss.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>
```

Note: `forEachStatus.index` is absolute with respect to the underlying collection, array or other type. For example, if `forEachBegin` is `5`, then the first value of `forEachStatus.index` with be `5`.

page - `org.zkoss.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 `org.zkoss.zk.ui.Page` interface.

requestScope – `java.util.Map`

A map of custom attributes associated with the current execution. It is the same as `getAttributes` method in the `org.zkoss.zk.ui.Execution` interface.

self - `org.zkoss.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 - `org.zkoss.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 `org.zkoss.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.

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

spaceOwner - `org.zkoss.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.

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 addition, 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 `org.zkoss.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, `org.zkoss.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 `import` Directive

```
<?import uri="..."?>
```

It imports the component definitions and initiators defined in another ZUML page. In other words, it imports the `component` and `init` directives from the specified page.

A typical use is that you put a set of component definitions in one ZUML page, and then import it in other ZUML pages, such that they share the same set of component definitions, additional to the system default.

```
<!-- special.zul: Common Definitions -->
```

```
<?init zscript="/WEB-INF/macros/special.zs"?>
<?component name="special" macro-uri="/WEB-INF/macros/special.zuml" class="Special"?>
<?component name="another" macro-uri="/WEB-INF/macros/another.zuml"?>
```

where the `Special` class is assumed to be defined in `/WEB-INF/macros/special.zs`.

Then, other ZUML pages can share the same set of component definitions as follows.

```
<?import uri="special.zul"?>
...
<special/><!-- you can use the component defined in special.zul -->
```

Notes

- Unlike other directives, the import directives must be at the topmost level, i.e., at the the same level as the root element.
- The imported component definitions in the imported page are also imported. For example, if A imports B and B imports C, then A imports both C and B component definitions. If there is a name conflict, A overrides B, while B overrides C.
- Once the component definitions is imported, it won't be changed until the page is change, no matter the imported page is changed or not.

uri

[Required]

The URI of a ZUML page which the component definitions will be imported from.

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 `org.zkoss.zk.ui.Execution` interface.

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

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

class

[Optional]

A class name that must implement the `org.zkoss.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 `link` and `meta` Directives

```
<?link [href="uri"] [name0="value0"] [name1="value1"] [name2="value2"]?>
<?meta [name0="value0"] [name1="value1"] [name2="value2"]?>
```

These are so-called header elements in HTML. Currently only HTML-based clients (so-called browsers) support them.

Developers can specify whatever attributes with these header directives. ZK only encodes the URI of the `href` attribute (by use of the `encodeURL` method of the `Executions` class). ZK generates all other attributes directly to the client.

Notice that these header directives are effective only for the main ZUL page. In other words, they are ignored if a page is included by another pages or servlets. Also, they are ignored if the page is a `zhtml` file.

```
<?link rel="alternate" type="application/rss+xml" title="RSS feed"
 href="/rssfeed.php"?>
<?link rel="shortcut icon" type="image/x-icon" href="/favicon.ico"?>

<window title="My App">
```

```
  My content
</window>
```

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 `org.zkoss.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 `org.zkoss.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="org.zkoss.zkplus.spring.DelegatingVariableResolver"?>
```

class

[Optional]

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

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.zkoss.org/2005/zk`

```
<zk:attribute xmlns:zk="http://www.zkoss.org/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;)"/>
```

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.

forEachBegin

[Optional][Default: 0]

It is used with the `forEach` attribute to specify the starting offset when iterating a collection of objects. If not specified, it iterates from the first element, i.e., 0 is assumed.

forEachBegin

[Optional][Default: 0]

It is used with the `forEach` attribute to specify the index (starting from 0) that the iteration shall begin at. If not specified, the iteration begins at the first element, i.e., 0 is assumed.

If `forEachBegin` is greater than or equals to the number of elements, no iteration is performed.

forEachEnd

[Optional][Default: *the last element*]

It is used with the `forEach` attribute to specify the index (starting from 0) the iteration shall ends at (inclusive). If not specified, the iterations ends at the last element.

If forEachEnd is greater than or equals to the number of elements, the iteration ends at the last element.

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 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.

- If there is one "*" is specified in an URL or URI such as `/my*.css`, then "*" will be 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.
- 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.
- All other "*" are ignored.

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.

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

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

ZK Attributes

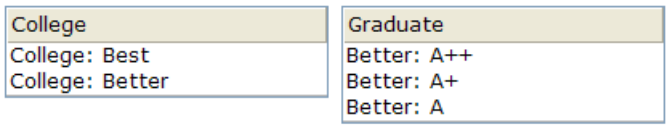
ZK attributes are used to control the associated element, other than initializing the data member.

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 `forEachBegin` Attribute

It is used with the `forEach` attribute to specify the index (starting from 0) that the iteration shall begin at. If not specified, the iteration begins at the first element, i.e., 0 is assumed.

If `forEachBegin` is greater than or equals to the number of elements, no iteration is performed.

Note: `forEachStatus.index` always starts from 0, no matter what `forEachBegin` is.

The `forEachEnd` Attribute

It is used with the `forEach` attribute to specify the index (starting from 0) the iteration shall ends at (inclusive). If not specified, the iterations ends at the last element.

If `forEachEnd` is greater than or equals to the number of elements, the iteration ends at the

last element.

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, `org.zkoss.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 (String, Object).

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, Object).

sessionScope - `java.util.Map`

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

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 `org.zkoss.zul.html` package.
- The XML name space is `http://www.zkoss.org/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 `org.zkoss.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

Splitter

Style

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 `org.zkoss.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 `org.zkoss.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 `org.zkoss.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 `org.zkoss.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 `org.zkoss.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, `org.zkoss.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

Appendix A. WEB-INF/web.xml

To add ZK a Web application, you have to add servlets, listeners and a optional filter to web.xml.

ZK Loader

[Required] Class: `org.zkoss.zk.ui.http.DHtmlLayoutServlet`

`DHtmlLayoutServlet` is a servlet used to load ZUML pages when the Web server receives URL requests sent by users.

It is suggested to map this servlet to the `zul` and `zhtml` extensions as shown in the **Sample** section below. It is OK if you want to map `xul` and `html`, too.

The Initial Parameters

init-param	Descriptions
update-uri	[Required] It specifies the URI which the ZK AU engine is mapped to. For example, if the ZK AU engine is mapped to `/zkau/*`, by use of `servlet-mapping`, then specify `/zkau` for this parameter. Note: if the servlet container is used with other Web server, like Apache, you have to map this update URI to the servlet container (in additions to `zul` and `zhtml` files).

ZK AU Engine

[Required] Class: `org.zkoss.zk.au.http.DHtmlUpdateServlet`

`DHtmlUpdateServlet` is a servlet that handles AJAX requests asynchronously and automatically.

Notice that the URL pattern mapped to this engine must be consistent with the `update-uri` parameter of the ZK Loader.

ZK Session Cleaner

[Required] Class: `org.zkoss.zk.ui.http.HttpSessionListener`

`HttpSessionListener` is a listener used to clean up memory when a HTTP session is destroyed.

ZK Filter

[Optional] Class: `org.zkoss.zk.ui.http.DHtmlLayoutFilter`

`DHtmlLayoutFilter` is a filter to post-process the responses generated by other servlets. Its role is similar to the ZK Loader. Unlike the ZK Loader, which loads static ZUML pages from Web applications directly, the ZK filter is designed to process dynamic pages generated by other servlets, say JSP or JSF. It enables developers to add rich user interfaces to existent servlets written in any technology.

The Initial Parameters

init-param	Descriptions
`extension`	[Optional][Default: `html`] It specifies how to process the response generated by other servlets. If `html` or `zhtml`, XHTML is assumed to be the default namespace. If `xul` or `zul`, XUL is assumed to be the default namespace.
`charset`	[Optional][Default: `UTF-8`] It specifies the default charset for the output of this filter. If an empty string is specified as follows, the container's default is used. In other words, the `setCharacterEncoding` method of `javax.servlet.ServletResponse` is not called. `<param-value></param-value>`

How to Specify in web.xml

```
<filter>
   <filter-name>zkFilter</filter-name>
   <filter-class>org.zkoss.zk.ui.http.DHtmlLayoutFilter</filter-class>
</filter>
```

Sample of web.xml

```
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

   <!-- //// -->
   <!-- ZK -->
   <listener>
      <description>Used to cleanup when a session is destroyed</description>
      <display-name>ZK Session Cleaner</display-name>
      <listener-class>org.zkoss.zk.ui.http.HttpSessionListener</listener-class>
   </listener>
   <servlet>
      <description>ZK loader for evaluating ZUML pages</description>
      <servlet-name>zkLoader</servlet-name>
      <servlet-class>org.zkoss.zk.ui.http.DHtmlLayoutServlet</servlet-class>

      <!-- Must. Specifies URI of the update engine (DHtmlUpdateServlet).
      It must be the same as <url-pattern> for the update engine.
      -->
      <init-param>
         <param-name>update-uri</param-name>
         <param-value>/zkau</param-value>
      </init-param>
      <load-on-startup>1</load-on-startup><!-- MUST -->
   </servlet>
   <servlet-mapping>
      <servlet-name>zkLoader</servlet-name>
      <url-pattern>*.zul</url-pattern>
   </servlet-mapping>
   <servlet-mapping>
      <servlet-name>zkLoader</servlet-name>
      <url-pattern>*.zhtml</url-pattern>
   </servlet-mapping>
   <servlet>
      <description>The asynchronous update engine for ZK</description>
      <servlet-name>auEngine</servlet-name>
      <servlet-class>org.zkoss.zk.au.http.DHtmlUpdateServlet</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>auEngine</servlet-name>
      <url-pattern>/zkau/*</url-pattern>
   </servlet-mapping>
   <!-- //// -->

   <!-- MIME mapping -->
   <mime-mapping>
      <extension>gif</extension>
      <mime-type>image/gif</mime-type>
```

```
   </mime-mapping>
   <mime-mapping>
      <extension>html</extension>
      <mime-type>text/html</mime-type>
   </mime-mapping>
   <mime-mapping>
      <extension>htm</extension>
      <mime-type>text/html</mime-type>
   </mime-mapping>
   <mime-mapping>
      <extension>jpeg</extension>
      <mime-type>image/jpeg</mime-type>
   </mime-mapping>
   <mime-mapping>
      <extension>jpg</extension>
      <mime-type>image/jpeg</mime-type>
   </mime-mapping>
   <mime-mapping>
      <extension>js</extension>
      <mime-type>application/x-javascript</mime-type>
   </mime-mapping>
   <mime-mapping>
      <extension>png</extension>
      <mime-type>image/png</mime-type>
   </mime-mapping>
   <mime-mapping>
      <extension>txt</extension>
      <mime-type>text/plain</mime-type>
   </mime-mapping>
   <mime-mapping>
      <extension>xml</extension>
      <mime-type>text/xml</mime-type>
   </mime-mapping>
   <mime-mapping>
      <extension>zhtml</extension>
      <mime-type>text/html</mime-type>
   </mime-mapping>
   <mime-mapping>
      <extension>zul</extension>
      <mime-type>text/html</mime-type>
   </mime-mapping>

   <welcome-file-list>
      <welcome-file>index.zul</welcome-file>
      <welcome-file>index.zhtml</welcome-file>
      <welcome-file>index.html</welcome-file>
      <welcome-file>index.htm</welcome-file>
   </welcome-file-list>
</web-app>
```

Appendix B. WEB-INF/zk.xml

`WEB-INF/zk.xml` is the configuration descriptor of ZK. This file optional. If you need to configure ZK differently from the default, you could provide a file called `zk.xml` under the `WEB-INF` directory.

Overview

The root element must be `<zk>`. Then, you could specify any combination of the following element under the root element.

The `listener` Element

You could specify any number of `listener` elements. Each of them could have two child elements, `description` and `listener-class`, where description is optional.

```
<zk>
   <listener>
      <listener-class>my.MyInit</listener-class>
   </listener>
</zk>
```

The type of a listener depends on what interface it implements. For example, if a listener implements the `org.zkoss.zk.ui.event.EventThreadInit` interface, then it is used to listen when an event processing thread is initialized. A listener could implement multiple interfaces and it will be used whenever the corresponding interface is about to call.

The `org.zkoss.zk.ui.event.EventThreadInit` Interface

It is implemented by a listener class that will be used to initialize an event processing thread, before an event is dispatched to it for processing.

If a listener implements this interface, an instance is created, and then the `prepare` method is called in the main thread (aka., the servlet thread), before processing an event. Then, the `init` method is called in the event processing thread.

If a developer wants to prevent an event from being processed, he can throw an exception in the `prepare` method or the `init` method.

A typical use of this feature is to implement auto-authentication. For example, JBoss[5] required you to call `SecurityAssociation.setPrincipal` to grant permissions of a user to the event processing thread, as described in the **Initialization Before Processing Each Event** section, the **Event Listening and Processing** chapter.

5 http://www.jboss.org

The `org.zkoss.zk.ui.event.EventThreadCleanup` interface

It is implemented by a listener class that will be used to cleanup an event processing thread, after it has processed an event.

If a listener implements this interface, an instance is created, and then the `cleanup` method is called in the event processing thread after the thread processes the event. Then, the `complete` method is called in the main thread (aka., the servlet thread), after the main thread is resumed.

> **Note**: The `complete` method won't be called if the corresponding `cleanup` method threw an exception.

A typical use of this feature is to clean up unclosed transaction.

Once registered, an instance is constructed and the `cleanup` method is called after leaving the event processing thread.

The `org.zkoss.zk.ui.event.EventThreadSuspend` interface

It is implemented by a listener class that will be called before an event processing thread is going to be suspended.

If a listener implements this interface, an instance is created, and then the `beforeSuspend` method, when an event processing thread is going to suspended. It executes in the event processing thread.

A developer can prevent can prevent an event processing thread from being suspended by throwing an exception.

A typical use of this feature is to limit the number of suspended threads.

The `org.zkoss.zk.ui.event.EventThreadResume` interface

It is implemented by a listener class that will be called after an event processing thread is resumed or aborted.

If a listener implements this interface, an instance is created, and then the `beforeResume` method is called in the main thread (aka., the servlet thread), when a suspended event thread is being resumed. Then, the `afterResume` method is called in the event processing thread after the thread is resumed successfully.

If a developer wants to prevent an event from being resumed, he can throw an exception in the `beforeResume` method.

Notice that `beforeResume` executes in the main thread, so it shares the same thread-local storage with the main thread. On the other hand, `afterResume` executes in the event processing thread, so it shares the same thread-local storage with the event thread (and application event listeners).

In additions to resuming normally, a suspended event processing thread might be aborted abnormally. For example, when the desktop is being destroyed, all suspended event threads will be aborted. When the suspended event processing thread is aborted, an instance is created, and the `abortResume` method is called in the main thread.

> **Note**: If a suspended event thread is aborted, none of the `beforeResume` and `afterResume` is called. Moreover, the `cleanup` and `complete` methods of `EventThreadCleanup` won't be called, either. Thus, you have to handle all necessary cleanups in `abortResume`.

The `org.zkoss.zk.ui.util.WebAppInit` interface

It is implemented by a listener class that will be used to initialize a ZK application.

When a ZK application is created, it invokes the `init` method of this interface such that developers could plug the application-specific codes to initialize the application.

The `org.zkoss.zk.ui.util.WebAppCleanup` interface

It is implemented by a listener class that will be used to cleanup a ZK application that is being destroyed.

When a ZK application is going to be destroyed, it invokes the `cleanup` method of this interface such that developers could plug the application-specific codes to cleanup the application.

The `org.zkoss.zk.ui.util.SessionInit` interface

It is implemented by a listener class that will be used to initialize a new session.

When ZK Loader created a new session, it invokes the `init` method of this interface such that developers could plug the application-specific codes to initialize a session.

A developer can prevent a session from being created by throwing an exception in the `init` method.

The `org.zkoss.zk.ui.util.SessionCleanup` interface

It is implemented by a listener class that will be used to cleanup a session that is being destroyed.

When ZK Loader is going to destroy a session, it invokes the `cleanup` method of this interface such that developers could plug the application-specific codes to cleanup a session.

The `org.zkoss.zk.ui.util.DesktopInit` interface

It is implemented by a listener class that will be used to initialize a new desktop.

When ZK Loader created a new desktop, it invokes the `init` method of this interface such that developers could plug the application-specific codes to initialize a desktop.

A developer can prevent a desktop from being created by throwing an exception in the `init` method.

The `org.zkoss.zk.ui.util.DesktopCleanup` interface

It is implemented by a listener class that will be used to cleanup a desktop that is being destroyed.

When ZK Loader is going to destroy a desktop, it invokes the `cleanup` method of this interface such that developers could plug the application-specific codes to cleanup a desktop.

The `org.zkoss.zk.ui.util.ExecutionInit` interface

It is implemented by a listener class that will be used to initialize a new execution.

When ZK Loader and Update Engine created a new execution, it invokes the `init` method of this interface such that developers could plug the application-specific codes to initialize an execution.

> **Tip**: Executions might be stacked. To know whether it is the first execution since a (Servlet) request is processed, you can check whether the `parent` argument is `null`.

A developer can prevent an execution from being created by throwing an exception in the `init` method.

The `org.zkoss.zk.ui.util.ExecutionCleanup` interface

It is implemented by a listener class that will be used to cleanup an execution that is being destroyed.

When ZK Loader is going to destroy an execution, it invokes the `cleanup` method of this interface such that developers could plug the application-specific codes to cleanup an execution.

The `org.zkoss.zk.ui.util.Monitor` interface

It is implemented by a listener class that will be used to monitor the statuses of ZK. Unlike other listener, there is at most one monitor listener for each Web application.

ZK provides an implementation named `org.zkoss.zk.ui.util.Statistic`, which accumulates the statistic data in the memory. It is a good starting point to understand the load of your ZK application.

The `log` Element

By default, ZK's logger depends on how the Web server is configured. However, you could configure ZK to load and monitor i3-log.conf as described in the **Logger** section of the **Beyond ZK** chapter.

```
<log>
   <log-base>org.zkoss</log-base>
</log>
```

If you want to use the same logging mechanism in your application, you could configure ZK to handle all loggers as follows.

```
<log>
   <log-base></log-base>
</log>
```

where an empty string means all packages, not just `org.zkoss` in the previous example.

The `desktop-config` Element

The allowed child elements include `theme-uri`, `desktop-timeout` and `file-check-period`. At most one `desktop-config` element is allowed for each `zk.xml`.

```
<desktop-config>
   <theme-uri>/my/blue**.css</theme-uri>
   <desktop-timeout>3600</desktop-timeout>
   <file-check-period>5</file-check-period>
</desktop-config>
```

The `theme-uri` Element

[Default: *none*]

It specifies the URI of an addition theme (aka., style sheets).

Like other URI, it accepts "*" for loading browser and Locale dependent style sheet. Refer to the **Browser and Locale Dependent URI** section in the **Internationalization** chapter for details.

```
<zk>
   <desktop-config>
      <theme-uri>/my/blue**.css</theme-uri>
   </desktop-config>
</zk>
```

Notice:

1. All style sheets defined in `lang.xml` and `lang-addon.xml` are loaded, no matter this parameter is defined or not. It is convenient for developers to override certain styles.

2. Each JAR could specify a `lang-addon.xml` file (under the `metainfo/zk` directory), so you could specify style sheets there if you have more than one style sheets.
3. You could specify extra CSS files for individual ZUML pages by use of the `style` component. Refer to the **ZUML with the XUL Component Set** chapter.

The `desktop-timeout` Element

[Default: 3600]

It specifies the time, in seconds, between client requests before a desktop is invalidated. A negative time indicates the session should never timeout.

The `file-check-period` Element

[Default: 5]

It specifies the time, in seconds, to wait before checking whether a file is modified.

For better performance, ZK has employed a cache to store parsed ZUML file. The time specified here controls how often ZK checks whether a file is modified. The larger the number the better the performance.

The `session-config` Element

The allowed child elements include `session-timeout` and `max-desktops-per-session`. At most one `session-config` element is allowed for each `zk.xml`.

```
<session-config>
   <timeout-uri>/my-timeout.zul</timeout-uri>
   <session-timeout>1800</session-timeout>
   <max-desktops-per-session>10</max-desktops-per-session>
</session-config>
```

The `timeout-uri` Element

[Default: *null*]

It specifies the target URI that will be used to redirect users to, when the desktop no longer exists – it is usually caused by session timeout. If this element is omitted, an error message will be shown up at the browser to alert users for what happens.

To reload the same URI again, you can specify an *empty* content as follows.

```
<session-timeout></session-timeout>
```

The `session-timeout` Element

[Default: *depending on the Web server*]

It specifies the time, in seconds, between client requests before a session is invalidated. A

negative time indicates the session should never timeout.

The `max-desktops-per-session` Element

[Default: 10]

It specifies the maximal allowed number of desktops per session. A desktop represents a HTML page for a browser. In other words, this number controls the number of concurrent browser windows allowed per session.

The `language-config` Element

The allowed child elements include `addon-uri`. At most one `language-config` element is allowed for each `zk.xml`.

```
<language-config>
   <addon-uri>/WEB-INF/lang-addon.xml</addon-uri>
   <addon-uri>/WEB-INF/lang-addon2.xml</addon-uri>
</language-config>
```

The `addon-uri` Element

[Default: *none*]

It specifies the URI of language add-on definitions. To specify more than one URIs, you have to define them with multiple `addon-uri`.

A language addon is used to add new components and override the definitions of existent components. Refer to **the Component Development Guide**.

The `system-config` Element

The allowed child elements include `max-event-threads`. At most one `system-config` element is allowed for each `zk.xml`.

```
<system-config>
   <max-event-threads>100</max-event-threads>
   <max-upload-size>5120</max-upload-size>
   <response-charset>UTF-8</response-charset>
   <locale-provider-class>my.LocaleProvider</locale-provider-class>
   <timeZone-provider-class>my.TimeZoneProvider</timeZone-provider-class>
   <cache-provider-class>my.CacheProvider</cache-provider-class>
   <ui-factory-class>my.UiFactory</ui-factory-class>
   <engine-class>my.UiEngine</engine-class>
</system-config>
```

The `max-event-threads` Element

[Default: 100]

It specifies the maximal allowed number of event handling threads. ZK will reuse the event processing threads until it exceeds the number specified here.

The `max-upload-size` Element

[Default: 5120]

It specifies the maximal allowed size, in kilobytes, to upload a file from the client.

The `response-charset` Element

[Default: UTF-8]

It specifies the charset for the rendering result of a ZUML page. In other words, it is used to load the ZUML page by the ZK Loader (i.e., DHtmlLayoutServlet).

If you want to use the container's default value, you can specify an empty string as follows.

```
<response-charset></response-charset>
```

The `locale-provider-class` Element

[Default: *null*]

It specifies which class used to determine the locale for a given session. The class must have a default constructor (without any argument), and implement the `org.zkoss.zk.ui.sys.LocaleProvider` interface.

The `timeZone-provider-class` Element

[Default: *null*]

It specifies which class used to determine the time zone for a given session. The class must have a default constructor (without any argument), and implement the `org.zkoss.zk.ui.sys.TimeZoneProvider` interface.

The `cache-provider-class` Element

[Default: `org.zkoss.zk.ui.impl.SessionDesktopCacheProvider`]

It specifies which class used to implement the desktop cache. The class must have a default constructor (without any argument), and implement the `org.zkoss.zk.ui.sys.DesktopCacheProvider` interface.

One instance of the cache provider is created and shared for each Web application, so you have to synchronize the access properly.

Available implementations are as follows.

Class	Description
org.zkoss.zk.ui.impl. SessionDesktopCacheProvider	It stores all desktops from the same session in one single cache. It is simple and fast, but not supporting clustering.
org.zkoss.zk.ui.impl. GlobalDesktopCacheProvider	It stores all desktops from the same Web application in one single cache. In other words, it doesn't count on session at all. It is useful because some Web server, e.g, BEA WebLogic[6], might be configured to use independent sessions for each request.

The `ui-factory-class` Element

[Default: `org.zkoss.zk.ui.http.SimpleUiFactory`]

It specifies which class used to create desktops and pages, and to convert URL to a page definition. The class must have a default constructor (without any argument), and implement the `org.zkoss.zk.ui.sys.UiFactory` interface.

One instance of the UI factory is created and shared for each Web application, so you have to synchronize the access properly.

A common use is to load page definitions and other UI information from the database, rather than from the resources of the Web application.

In addition, you might use it to implement a controller in a MVC model, such that it creates the correct desktop based on the request URL.

Available implementations are as follows.

Class	Description
org.zkoss.zk.ui.http. SimpleUiFactory	The default UI factory. The sessions generated by this factory is *not* serializable
org.zkoss.zk.ui.http. SerializableUiFactory	The sessions generated by this factory is serializable. If you want to store sessions when the Web server is shutdown and restore them after it started, you can specify this implementation.

The `engine-class` Element

[Default: `org.zkoss.zk.ui.impl.UiEngineImpl`]

It specifies which class used to implement the UI Engine. The class must have a default constructor (without any argument), and implement the `org.zkoss.zk.ui.sys.UiEngine` interface.

6 http://www.bea.com

One instance of the UI egnine is created and shared for each Web application, so you have to synchronize the access properly.

The `el-config` Element

The allowed child elements include `evaluator-class`. At most one `el-config` element is allowed for each `zk.xml`.

```
<el-config>
   <evaluator-class>my.MyExpressionEvaluatorImpl</evaluator-class>
</el-config>
```

The `evaluator-class` Element

[Default: `org.apache.commons.el.ExpressionEvaluatorImpl`]

It specifies the class used to evaluate EL expressions. If not specified, ZK uses the EL implementation from the Apache group, `org.apache.commons.el.ExpressionEvaluatorImpl`. If your Web server uses another implementation, you have to specify a proper class here.

The `error-page` Element

```
<error-page>
   <exception-type>ClassName</exception-type>
   <location>the error page's URI</location>
</error-page>
```

It specifies an error page used when an un-caught exception is thrown in updating a ZUML page (e.g., in an event listener). Each page is associated with an exception type, aka, a class deriving from `java.lang.Throwable`. You can specify multiple error pages, each with a different exception type. When an error occurs, ZK searches the proper error page by examing the exception type one-by-one. If none is found, it shows, by default, an alert message at the client.

The error page's root element must be a `window` component. It becomes a modal dialog once shown up at the client.

The `preference` Element

```
<preference>
   <name>any name</name>
   <value>any value</value>
</preference>
```

You can specify any number of preference with the `preference` element depicted above. The name and value are application specific and you can specify whatever value you like. To avoid name conflict, it is suggested to prefix the name with your domain name, such as

com.poitx.some.another.

The preferences can then be retrieved back by calling the `getPreference` method of the `org.zkoss.zk.ui.util.Configuration` class. Notice that each Web application has one configuration, which can be found by use of `getConfiguration` method of the `org.zkoss.zk.ui.WebApp` interface.

```
String value = webApp.getConfiguration().getPreference("org.zkoss.name", null);
if (value != null) {
   ...
}
```