



PRELIMINARY

SIMPLY RICH

ZK™

The Developer's Reference

Version 2.5.0

August 2007

Potix Corporation

Revision 68

Copyright © Potix Corporation. All rights reserved.

The material in this document is for information only and is subject to change without notice. While reasonable efforts have been made to assure its accuracy, Potix Corporation assumes no liability resulting from errors or omissions in this document, or from the use of the information contained herein.

Potix Corporation may have patents, patent applications, copyright or other intellectual property rights covering the subject matter of this document. The furnishing of this document does not give you any license to these patents, copyrights or other intellectual property.

Potix Corporation reserves the right to make changes in the product design without reservation and without notification to its users.

The Potix logo and ZK are trademarks of Potix Corporation.

All other product names are trademarks, registered trademarks, or trade names of their respective owners.

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.....	20
The variable-resolver Directive.....	21
ZK Elements.....	22
The XML Namespace.....	22
The attribute Element.....	22
The custom-attributes Element.....	23
The variables Element.....	24
The zk Element.....	24
The zscript Element.....	26
ZK Attributes.....	28
The forEach Attribute.....	28

The forEachBegin Attribute.....	29
The forEachEnd Attribute.....	29
The forward Attribute.....	29
The fulfill Attribute.....	30
The if Attribute.....	30
The unless Attribute.....	30
The use Attribute.....	30

3. EL Expressions.....31

Overview.....	31
Using EL Expressions.....	31
Variables.....	31
Implicit Objects.....	31
Literals.....	31
Operators.....	31
Functions.....	31
Standard Implicit Objects that ZK supports.....	32
applicationScope - java.util.Map.....	32
cookie - java.util.Map.....	32
header - java.util.Map.....	32
headerValues - java.util.Map.....	32
pageContext - javax.servlet.jsp.PageContext.....	32
pageScope - java.util.Map.....	32
param - java.util.Map.....	32
paramValues - java.util.Map.....	33
requestScope - java.util.Map.....	33
sessionScope - java.util.Map.....	33
ZK Implicit Objects.....	33

4. The XUL Components.....34

Overview.....	34
XulElement.....	34
Components.....	34
Audio.....	34
Box.....	34
Button.....	34
Caption.....	34
Checkbox.....	34
Column.....	34

Columns.....	35
Combobox.....	35
Comboitem.....	35
Datebox.....	35
Decimalbox.....	35
Div.....	35
Grid.....	35
Groupbox.....	35
Hbox.....	35
Html.....	35
Iframe.....	35
Image.....	35
Include.....	36
Intbox.....	36
Label.....	36
Listbox.....	36
Listcell.....	36
Listfoot.....	36
Listfooter.....	36
Listhead.....	36
Listheader.....	36
Listitem.....	36
Menu.....	36
Menubar.....	36
MenuItem.....	37
Menupopup.....	37
Menuseparator.....	37
Popup.....	37
Popupset.....	37
Radio.....	37
Radiogroup.....	37
Row.....	37
Rows.....	37
Separator.....	37
Slider.....	37
Space.....	37
Splitter.....	38
Style.....	38
Tab.....	38
Tabbox.....	38

Tabpanel.....	38
Tabpanels.....	38
Tabs.....	38
Textbox.....	38
Timer.....	38
Toolbar.....	38
Toobarbutton.....	38
Tree.....	38
Treecell.....	39
Treecchildren.....	39
Treecol.....	39
Treecols.....	39
Treeitem.....	39
Treerow.....	39
Vbox.....	39
Window.....	39
Supplemental Classes.....	39
AbstractListModel.....	39
Constraint.....	39
Constrained.....	39
Fileupload.....	40
ListModel.....	40
ListitemRenderer.....	40
MessageBox.....	40
RendererCtrl.....	40
SimpleConstraint.....	40
SimpleListModel.....	40
5. The XHTML Components.....	41
Overview.....	41
URL and encodeURL.....	41
AbstractTag.....	41
Raw.....	42
Components.....	42
A.....	42
Abbr.....	42
Acronym.....	42
Address.....	42
Area.....	42

B.....	42
Base.....	43
Big.....	43
Blockquote.....	43
Body.....	43
Br.....	43
Button.....	43
Caption.....	43
Cite.....	43
Code.....	43
Collection.....	43
Colgroup.....	43
Dd.....	43
Del.....	44
Dfn.....	44
Dir.....	44
Div.....	44
Dl.....	44
Dt.....	44
Em.....	44
Embed.....	44
Fieldset.....	44
Font.....	44
Form.....	44
H1.....	44
H2.....	45
H3.....	45
H4.....	45
Head.....	45
Hr.....	45
Html.....	45
I.....	45
Iframe.....	45
Img.....	45
Input.....	45
Ins.....	45
Isindex.....	45
Kbd.....	46
Label.....	46
Legend.....	46

Li.....	46
Link.....	46
Map.....	46
Menu.....	46
Meta.....	46
Nobr.....	46
Object.....	46
Ol.....	46
Optgroup.....	46
Option.....	47
P.....	47
Pre.....	47
Q.....	47
S.....	47
Sam.....	47
Script.....	47
Select.....	47
Small.....	47
Span.....	47
Strong.....	47
Style.....	47
Sub.....	48
Sup.....	48
Table.....	48
Tbody.....	48
Td.....	48
Text.....	48
Textarea.....	48
Tfoot.....	48
Th.....	48
Thead.....	48
Title.....	48
Tr.....	48
Tt.....	49
Ul.....	49
Var.....	49
Supplement Classes.....	49
Fileupload.....	49
MessageBox.....	49

Appendix A. WEB-INF/web.xml.....	50
ZK Loader.....	50
The Initial Parameters.....	50
ZK AU Engine.....	51
ZK Session Cleaner.....	51
ZK Filter.....	51
The Initial Parameters.....	51
How to Specify in web.xml.....	52
DSP Loader.....	52
The Initial Parameters.....	52
How to Specify in web.xml.....	53
Sample of web.xml.....	53
Appendix B. WEB-INF/zk.xml.....	56
Overview.....	56
The richlet and richlet-mapping elements.....	56
The listener Element.....	57
The log Element.....	61
The client-config Element.....	62
The desktop-config Element.....	63
The el-config Element.....	65
The language-config Element.....	65
The session-config Element.....	66
The system-config Element.....	67
The zscript-config Element.....	71
The device-config Element.....	72
The error-page Element.....	73
The preference Element.....	73

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., zscript) 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` will 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 to 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" [inline="true|false"]
  [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 `inline` attribute specifies whether it is an inline macro (`inline="true"`) or a regular macro (default).

An inline macro behaves like *inline-expansion*. ZK doesn't create a macro component if an inline macro is encountered. Rather, it inline-expands the components defined in the macro URI. In other words, it works as if you type the content of the inline macro directly to the target page.

On the other hand, ZK will create a real component (called a macro component) to represent the regular macro. That is, the macro component is created as the parent of the components that are defined in the macro.

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][EL is *not* allowed]

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][EL is allowed]

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 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.Initiator` interface.

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

class

[Optional]

A class name that must implement the `org.zkoss.zk.ui.util.Initiator` 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"]
[zscript-language="Java"] [xmlns:any-prefix="any-url"?>
```

It describes attributes of a page.

id

[Optional][Default: *generated automatically*][EL allowed]

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*][EL allowed]

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%`][EL allowed]

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*][Allowed values: `xul/html` | `xhtml`]

Specifies the markup language for this page. The markup language determines the

default component set. Currently, it supports `xul/html` and `xhtml`.

Note: You can place the page directive in any location of a XML document, but the `language` attribute is meaningful only if the directive is located at the topmost level.

zscript-language

[Optional][Default: `Java`][Allowed values: `Java` | `JavaScript` | `Ruby` | `Groovy`]

Specifies the default scripting language, which is assumed if an `zscript` element doesn't specify any scripting language explicitly.

```
<?page zscript-language="JavaScript"?>

<zscript>
    var m = round(box.value); //JavaScript is assumed.
</zscript>
```

If this option is omitted, `Java` is assumed. Currently ZK supports four different languages: `Java`, `JavaScript`, `Ruby` and `Groovy`. This option is case insensitive.

Note: Deployers can extend the number of supported scripting languages. Refer to the **How to Support More Scripting Language** section in **the Developer's Guide**.

xmlns:any-prefix

[Optional][Default: *none*][EL allowed]

Specifies the XML namespaces that will be generated to the output. For Ajax devices, it becomes the attributes of the `html` element of the generated output.

```
<?page xmlns:v="urn:schemas-microsoft-com:vml" ?>
```

generates the HTML output as follows

```
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:v="urn:schemas-microsoft-com:vml">
```

Note: `xmlns="http://www.w3.org/1999/xhtml"` is always generated.

Note: If the value is specified with an EL expression and it is evaluated to null, the corresponding attribute won't be generated.

The taglib Directive

```
<?taglib 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 the built-in TLD files identified as

`http://www.zkoss.org/dsp/web/core` and then use one of these function called 1.

```
<?taglib uri="http://www.zkoss.org/dsp/web/core" prefix="c"?>
<window title="{c:l('my.title')}}">
...
</window>
```

Tip: ZK searches all TLD files defined in the `/META-INF/tld/config.xml` file from the classpath. If you want ZK to load your custom TLD files, add them to class path and then specify the following content in the `/META-INF/tld/config.xml` file.

```
<taglib>
  <taglib-uri>http://your-domain.com/your-path</taglib-uri>
  <taglib-location>/the/path/of/your/tld/file</taglib-location>
</taglib>
```

If you to load a TLD file from your Web application, you can specify the path as follows.

```
<?taglib uri="/WEB-INF/tld/my.tld" prefix="my"?>
```

uri

[Required][EL is *not* allowed]

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.scripting.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.scripting.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" [trim="true|false"]>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('&quot;Hi&quot;');"/>
```

Another example:

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

name

[Required]

Specifies the attribute name.

trim

[Optional][Default: false]

Specifies whether to omit the leading and trailing whitespaces of the attribute value.

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 retrieved 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 **variables** Element

```
<variables [local="false|true] var1="value1" [var2="value2"...]/>
```

It defines a set of variables for the ID space it belongs. It is equivalent to the `setVariable` method of `Component`, if it has a parent component, and `Page`, if it is declared at the page level.

You could specify as many as variables you want. These variables are stored to the namespace of the ID space it belongs. Thus, they can be accessible by the interpreters and EL expressions.

```
<variables cd="{param.cd}" less="more"/>
```

local

[optional][Default: `false`]

Specifies whether to store the variable always at the current ID space. By default, it is `false`. It means ZK will check the existence of any variable with the same name by looking up the current ID space, the parent ID space, and parent's parent, and so on. If found, the variable's value is replaced with the value specified here. If not, a local variable is created. If `true` is specified, it doesn't look up any parent ID space.

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 [language="Java|JavaScript|Ruby|Groovy"]>Scripting codes</zscript>
<zscript src="uri" [language="Java|JavaScript|Ruby|Groovy"]/>
```

It defines a piece of scripting codes that will be interpreted when the page is evaluated. The language of the scripting codes is, by default, Java. You can select a different language by use the `language` attribute².

The `zscript` element has two formats as shown above. The first format is used to embed the scripting codes directly in the page. The second format is used to reference an external file that contains the scripting codes.

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

² Furthermore, you can use the page directive to change the default scripting language other than Java.

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 the scripting codes. If specified, the scripting codes will be loaded as if they are embedded directly.

Note: the file shall contain the source codes in the selected scripting language. The encoding must be UTF-8. 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 content³.
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 browsers⁴.
If the last "*" will be replaced with a proper Locale as described above.
- All other "*" are ignored.

language

[Optional][Default: the page's default scripting language]

[Allowed Values: Java | JavaScript | Ruby | Groovy]

It specifies the scripting language which the scripting codes are written in.

³ Refer to the `getContext` meth from the `javax.servlet.ServletContext` interface.

⁴ In the future editions, we will use different codes for browsers other than IE and FF.

deferred

[Optional][Default: `false`]

Specifies whether to defer the evaluation of this element until the first non-deferred `zscript` codes of the same language has to be evaluated. It is used to defer the loading of the interpreter and then speed up the loading of a ZUML page. For example, if all `zscript` elements are deferred, they are evaluated only when the first event listened by a handler implemented in `zscript` is received.

Refer to the **How to Defer the Evaluation** section in the **Developer's Guide**.

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

```
forEach="{an-EL-expr}"
```

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]}/>
  </listitem>
</listbox>
</hbox>

```

College	Graduate
College: Best	Better: A++
College: Better	Better: A+
	Better: A

The `forEachBegin` Attribute

```
forEachBegin="{an-EL-expr}"
```

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

```
forEachEnd="{an-EL-expr}"
```

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 `forward` Attribute

```
forward="originalEvent=targetId1/targetId2,targetEvent"
```

```
forward="originalEvent={el-expr},targetEvent"
```

```
forward="targetEvent"
```

It is used to forward an event, that is targeting a specific component, to another component in another event name. It is called the forward condition.

The original event is optional. If it is omitted, `onClick` is assumed. Similarly, the target ID is also optional. If omitted, the space owner is assumed.

If you want to forward several events, you can specify these conditions in the `forward` attribute by separating them with the comma (,):

```
<textbox forward="onChanging=onUpdating, onChange=some.onUpdate"/>
```

The `fulfill` Attribute

```
fulfill="event-name"  
fulfill="target-id.event-name"  
fulfill="id1/id2/id3.event-name"  
fulfill="${el-expr}.event-name"
```

It is used to specify when to create the child components. By default (i.e., `fulfill` is not specified), the child components are created right after its parent component, at the time the ZUML page is loaded.

If you want to defer the creation of the child components, you can specify the condition with the `fulfill` attribute. The condition consists of the event name and, optionally, the target component's identifier or path. It means that the child elements won't be processed, until the event is received by, if specified, the target component. If the identifier is omitted, the same component is assumed.

If an EL expression is specified, it must return a component, an identifier or a path.

The `if` Attribute

```
if="${an-EL-expr}"
```

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

```
unless="${an-EL-expr}"
```

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

```
forEachEnd="a-class-name"
```

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 **Implicit 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

Constrained

Fileupload

ListModel

ListitemRenderer

MessageBox

RendererCtrl

SimpleConstraint

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="http://www.zkoss.org/dsp/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 automatically⁵.

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 non-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`).

⁵ The reason not to handle XHTML components 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

DI

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

UI

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.

Notice that you must specify `load-on-startup` since many other servlets depend on the ZK loader.

```
<load-on-startup>1</load-on-startup>
```

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	<p>[Required]</p> <p>It specifies the URI which the ZK AU engine is mapped to.</p> <p>For example, if the ZK AU engine is mapped to <code>/zkau/*</code>, by use of <code>servlet-mapping</code>, then specify <code>/zkau</code> for this parameter.</p> <p>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 <code>zul</code> and <code>zhtml</code> files).</p>
compress	<p>[Optional][Default: <code>true</code>]</p> <p>It specifies whether to compress the output if the browser supports the compression (<code>Accept-Encoding</code>) and this Servlet is not included by other Servlets.</p>
log-level	<p>[Optional]</p> <p>It specifies the default log level for <code>org.zkoss</code>. If not specified, the system default (usually <code>INFO</code>) is used.</p> <p>Possible values: <code>OFF</code>, <code>ERROR</code>, <code>WARNING</code>, <code>INFO</code>, <code>DEBUG</code> and <code>FINER</code>. Refer to the Beyond ZK chapter in the Developer's Guide.</p>

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 output generated by other servlets, such as JSP pages. 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.

Note: the output must be in XHTML (or ZUML) syntax. If you encounter any problem, you can save the generated output into a ZHTML page and then browse the URL whether the ZHTML page is stored.

The Initial Parameters

init-param	Descriptions
extension	[Optional][Default: <code>html</code>] It specifies how to process the response generated by other servlets. If <code>html</code> or <code>zhtml</code> , XHTML is assumed to be the default namespace. If <code>xul</code> or <code>zul</code> , XUL is assumed to be the default namespace.
charset	[Optional][Default: <code>UTF-8</code>] 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 <code>setCharacterEncoding</code> method of

init-param	Descriptions
	<p><code>javax.servlet.ServletResponse</code> is not called.</p> <pre><param-value></param-value></pre>
compress	<p>[Optional][Default: true]</p> <p>It specifies whether to compress the output if the browser supports the compression (<code>Accept-Encoding</code>) and this filter is not included by other Servlets.</p>

How to Specify in web.xml

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

DSP Loader

[Optional] Class: `org.zkoss.web.servlet.dsp.InterpreterServlet`

`InterpreterServlet` is a servlet used to process the DSP files. DSP is a JSP-like template technology. It takes the same syntax as that of JSP. Unlike JSP, DSP is interpreted at the run time, so it is easy to deploy DSP pages. No Java compiler is required in your run-time environment. In addition, you could distribute DSP pages in jar files. This is the way ZK is distributed.

However, you cannot embed Java codes in DSP pages. Actions of DSP, though extensible through TLD files, are different from JSP tags.

The Initial Parameters

init-param	Descriptions
charset	<p>[Optional][Default: UTF-8]</p> <p>It specifies the default charset for the output of this filter.</p> <p>If an empty string is specified as follows, the container's default is used. In other words, the <code>setCharacterEncoding</code> method of <code>javax.servlet.ServletResponse</code> is not called.</p>
class-resource	<p>[Optional][Default: false]</p> <p>Specifies whether to load resources, such as TLD files, from the class loader, in addition to the servlet context.</p>
compress	<p>[Optional][Default: true]</p>

init-param	Descriptions
	It specifies whether to compress the output if the browser supports the compression (Accept-Encoding) and this Servlet is not included by other Servlets.

How to Specify in web.xml

```
<servlet>
  <servlet-name>zkLoader</servlet-name>
  <servlet-class>org.zkoss.web.servlet.dsp.InterpreterServlet</servlet-class>
</servlet>
```

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>jad</extension>
  <mime-type>text/vnd.sun.j2me.app-descriptor</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 `richlet` and `richlet-mapping` elements

To declare a `richlet`, you have to add the `richlet` element to `zk.xml`. You could specify any number of `richlet` elements. Each of them must have two child elements, `richlet-name` and `richlet-class`, and might have any number of the `init-param` child elements.

The class name specified in the `richlet-class` element must implement the `org.zkoss.zk.ui.Richlet` interface. The name and value specified in the `init-param` element can be retrieved when the `init` method of `org.zkoss.zk.ui.Richlet` is called.

```
<richlet>
  <richlet-name>Test</richlet-name>
  <richlet-class>org.zkoss.zkdemo.TestRichlet</richlet-class>
  <init-param>
    <param-name>any</param-name>
    <param-value>any</param-value>
  </init-param>
</richlet>
```

Once declaring a `richlet`, you can map it to any number of URL by use of `richlet-mapping` as depicted below.

```
<richlet-mapping>
  <richlet-name>Test</richlet-name>
  <url-pattern>/test</url-pattern>
</richlet-mapping>
<richlet-mapping>
  <richlet-name>Test</richlet-name>
  <url-pattern>/some/more/*</url-pattern>
</richlet-mapping>
```

The URL specified in the `url-pattern` element must start with `/`. If the URI ends with `/*`, then it is matched to all request with the same prefix. To retrieve the real request, you can check the value returned by the `getRequestPath` method of the current page.


```
public void service(Page page) {
    if ("/some/more/hi".equals(page.getRequestPath())) {
        ...
    }
}
```

The listener Element

To declare a listener, you have to add the `listener` element to `zk.xml`. 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, JBoss⁶ 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.

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.

⁶ <http://www.jboss.org>

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 be suspended. It executes in the event processing thread.

A developer 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.URIInterceptor` interface

It is implemented by a listener class that will be used to intercept the retrieving of ZUML pages with the associated URI. Once registered, an instance of the specified class is created, and then the `request` method is invoked, each time the application wants to retrieve the page definition of a page based on an URI.

A typical use of this interface is to ensure the current user has the authority to access the certain URI.

You can register any number of URI interceptors (`URIInterceptor`).

Note:

1. Unlike `ExecutionInit` and many other listeners, an instance of the registered `URIInterceptor` is created at the time of registration, and then it is shared by the whole application. Thus, you have to make sure it can be accessed concurrently.

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

It is implemented by a listener class that will be used to intercept each request made to ZK Loader and ZK Update Engine. Once registered, an instance of the specified class is created, and then the `request` method is invoked, each time a request is received by ZK Loader or ZK Update Engine.

A typical use of this interface is to determine the locale and/or time zone of the request. Refer to the **Developer's Guide** for more information.

You can register any number of the request interceptors (`RequestInterceptor`).

Note:

1. Unlike `ExecutionInit` and many other listeners, an instance of the registered `RequestInterceptor` is created at the time of registration, and then it is shared by the whole application. Thus, you have to make sure it can be accessed concurrently.
2. The request parameters will be parsed with the proper locale and character encoding, after the `request` method is called. It is not recommended to call the `getParameter` or `getParameterValues` methods (of `javax.servlet.ServletRequest`) in this method.

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 `client-config` Element

It is used to customize the behavior of the ZK Client Engine. You might have multiple `client-config` elements in one `zk.xml`.

```
<client-config>
  <disable-behind-modal>true</disable-behind-modal>
  <error-reload>
    <error-code>301</error-code>
    <reload-uri></reload-uri>
  </error-reload>
  <keep-across-visits>true</keep-across-visits>
  <processing-prompt-delay>900</processing-prompt-delay>
  <tooltip-delay>800</tooltip-delay>
</client-config>
```

The `disable-behind-modal` Element

[Default: `true`]

When a modal window is opened, all components that don't belong to the modal window are disabled. However, if the page is very big, the performance may not be acceptable (depending on the JavaScript interpreter of the browsers). If opening up a modal window is too slow for you, you can turn off this feature. The side effect is that a user may use TAB to move the focus to a component that shall be disabled.

The `error-reload` Element

[Default: *reload if 301, 402 or 403; show an error message, otherwise*]

It specifies what URI to redirect the browser to. For example, if you prefer to redirect to the login page, say, `login.zul`, you can specify the following in `zk.xml`:

```
<error-reload>
  <error-code>301</error-code>
  <reload-uri>/login.zul</reload-uri>
</error-reload>
<error-reload>
  <error-code>402</error-code>
  <reload-uri>/login.zul</reload-uri>
</error-reload>
<error-reload>
  <error-code>403</error-code>
  <reload-uri>/login.zul</reload-uri>
</error-reload>
```

If the content of `reload-uri` is empty, the browser simply reloads the same page again.

```
<reload-uri></reload-uri>
```

If you want to show an error message instead, specify `false`.

```
<reload-uri>false</reload-uri>
```

The `keep-across-visits` Element

[Default: `false`⁷]

It specifies whether to keep the desktop when a user reloads an URL or browses away to another URL. Since browsers won't cache HTML pages generated by ZK, ZK removes a desktop as soon as the user reloads the URL or browses to another URL.

However, you have to specify `keep-across-visits` with `true`, if you use the server-side cache for the HTML pages generated by ZK. An example of the server side cache is OpenSymphony `CacheFilter`⁸.

```
<client-config>
  <keep-across-visits>true</keep-across-visits>
</client-config>
```

The `processing-prompt-delay` Element

[Default: 900]

It specifies the time, in milliseconds, to wait before prompting the user with a dialog indicating that the request is in processing.

The `tooltip-delay` Element

[Default: 800]

It specifies the time, in milliseconds, to wait before popping up the tooltip when the user moves the mouse pointer over particular UI components.

The `desktop-config` Element

It is used to customize how ZK handles desktops. You might have multiple `desktop-config` elements in one `zk.xml`.

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

⁷ Exception: the Opera browser.

⁸ <http://www.opensymphony.com/oscache/wiki/CacheFilter.html>

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 desktop should never timeout.

The `disable-default-theme` Element

[Default: *none*]

It specifies the language name (aka., the component set) whose default theme shall be disabled. For example, the following statement disables the style sheet of the XUL component set (its language name is `xul/html`).

```
<desktop-config>
  <disable-default-theme>xul/html</disable-default-theme>
</desktop-config>
```

Notice that `theme-uri` adds additional style sheet files. It doesn't affect the default theme, unless `disable-default-theme` is specified.

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 `theme-uri` Element

[Default: *none*]

It specifies the URI of an addition theme (aka., a style sheet file).

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.

You can specify any number of `them-uri` as follows.

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

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 `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 `language-config` Element

The allowed child elements include `addon-uri`. You might have multiple `language-config` elements in one `zk.xml`.

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

Note: Unlike most other configurations defined in `WEB-INF/zk.xml`, the definitions defined in language addons are applied to all Web applications sharing the same `zk.jar`.

In other words, the definitions in language addons are visible to all Web applications sharing the same `zk.jar`. Furthermore, it may cause errors in another Web application, if the classes or resources are available only in the Web application defining this.

Thus, if it is an issue, just put `zk.jar` and relevant ZK libraries under the `WEB-INF/lib` directory.

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 `session-config` Element

The allowed child elements include `session-timeout` and `max-desktops-per-session`. You might have multiple `session-config` elements in one `zk.xml`.

```
<session-config>
  <session-timeout>1800</session-timeout>
  <timer-keep-alive>false</timer-keep-alive>
  <max-desktops-per-session>10</max-desktops-per-session>
</session-config>
```

The `session-timeout` Element

[Default: 0 (*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 default zero means to use the system default (which is usually specified in `web.xml`).

The `timer-keep-alive` Element

[Default: *false*]

It specifies whether to keep the session alive, when receiving the `onTimer` event.

A session is considered as timeout (and then invalidated), if it doesn't receive any client request in the specified timeout interval (see the **`session-timeout`** element above).

By setting this option to true, the `onTimer` event, just like any other events, will reset the session timeout counter (and then keep the session alive until timeout). Notice that, if this option is false and the timer is shorter than the session timeout, the session won't be expired.

By default, this option is false. It means the `onTimer` event is ignored when handling the session timeout. In other words, the session will expire if no other event is received before 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.

Note: If you use `org.zkoss.zk.ui.impl.GlobalDesktopCacheProvider`, then you have to make this number much larger since it means the maximal allowed number of desktops *per system*.

The `system-config` Element

You might have multiple `system-config` elements in one `zk.xml`.

```
<system-config>
  <cache-provider-class>my.CacheProvider</cache-provider-class>
  <disable-event-thread/>
  <engine-class>my.UiEngine</engine-class>
  <failover-manager-class>my.FailoverManager</failover-manager-class>
  <id-generator-class>my.IdGenerator</id-generator-class>
  <max-spare-threads>100</max-spare-threads>
  <max-suspended-threads>100</max-suspended-threads>
  <max-upload-size>5120</max-upload-size>
  <max-process-time>3000</max-process-time>
  <response-charset>UTF-8</response-charset>
  <server-push-class>my.ServerPush</server-push-class>
  <upload-charset>UTF-8</upload-charset>
  <ui-factory-class>my.UiFactory</ui-factory-class>
  <web-app-class>my.WebApp</web-app-class>
</system-config>
```

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
<code>org.zkoss.zk.ui.impl.SessionDesktopCacheProvider</code>	It stores all desktops from the same session in one single cache. It is simple and fast, but not supporting clustering.

Class	Description
<code>org.zkoss.zk.ui.impl. GlobalDesktopCacheProvider</code>	<p>It stores all desktops from the same Web application in one single cache. In other words, it doesn't count on session at all.</p> <p>It is useful because some Web server, e.g, BEA WebLogic⁹, might be configured to use independent sessions for each request.</p>

The `disable-event-thread` Element

[Default: false (enabled)]

It specifies whether to disable the use of the event processing thread. If disabled, no event processing thread will be used at all. In other words, all events are processed in the Servlet thread directly.

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.

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

The `failover-manager-class` Element

[Default: *none*]

It specifies which class used to handle the failover. It is called to recover a desktop, when ZK cannot locate a desktop. The class must have a default constructor (without any argument), and implement the `org.zkoss.zk.ui.sys.FailoverManager` interface.

In most cases, you don't need to provide any implementation. Rather, you can let Web servers to handle failover and clustering for you by specifying the `org.zkoss.zk.ui.http.SerializableUiFactory` class in the `ui-factory-class` element as described above.

The `id-generator-class` Element

[Default: *none*]

It specifies which class used to generate UUID of page and components, and ID of desktops. The class must have a default constructor (without any argument), and

⁹ <http://www.bea.com>

implement the `org.zkoss.zk.ui.sys.IdGenerator` interface.

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

If no ID generator is specified, the default ID generation algorithm will be used.

The `max-spare-threads` Element

[Default: 100]

It specifies the maximal allowed number of the thread pool for queuing the idle event processing threads. ZK will reuse the idle event processing threads by keeping them in a thread pool. The number specified here then controls the maximal size of the pool.

A negative value indicates there is no limit. Zero means no pool at all.

The `max-suspended-threads` Element

[Default: -1 (*no limit*)]

It specifies the maximal allowed number of the suspended event processing threads. A negative value indicates there is no limit at all.

An instance of `org.zkoss.zk.ui.TooManySuspendedException` is thrown, if an event processing thread is going to suspend and the number of suspended threads exceeds the number specified here. You can use the `error-page` element to control how to display this error, or catch the exception and handle it in a different way.

The `max-upload-size` Element

[Default: 5120]

It specifies the maximal allowed size, in kilobytes, to upload a file from the client. A negative value indicates there is no limit.

The `max-process-time` Element

[Default: 3000]

It specifies the maximal allowed time to process events, in milliseconds. It must be positive. ZK will keep processing the requests sent from the client until all requests are processed, or the maximal allowed time expires.

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 `server-push-class` Element

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

It specifies which class used to implement the server-push feature. The class must have a default constructor (without any argument), and implement the `org.zkoss.zk.ui.sys.ServerPush` interface.

The `upload-charset` Element

[Default: UTF-8]

It specifies the charset for the uploaded text files if the content type is not specified, or no charset is specified with the content type.

If the uploaded file is binary, there is no encoding issue at all.

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
<code>org.zkoss.zk.ui.http.SimpleUiFactory</code>	The default UI factory. The sessions generated by this factory is <i>not</i> serializable
<code>org.zkoss.zk.ui.http.SerializableUiFactory</code>	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 web-app-class Element

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

It specifies which class used to implement the Web application. The class must have a default constructor (without any argument), and implement both the `org.zkoss.zk.ui.WebApp` and `org.zkoss.zk.ui.sys.WebAppCtrl` interfaces. Instead of implementing from scratch, you can extend it from the `org.zkoss.zk.ui.impl.AbstractWebApp` or `org.zkoss.zk.ui.http.SimpleWebApp` classes.

The zscript-config Element

It configures the interpreters to interpret the zscript codes. The allowed child element is `language-name` and `interpreter-class`. You might have multiple `zscript-config` elements in one `zk.xml`.

```
<zscript-config>
  <language-name>Java</language-name><!-- case insensitive --!>
  <interpreter-class>my.MySuperJavaInterpreter</interpreter-class>
</zscript-config>
```

Note: Unlike most other configurations defined in `WEB-INF/zk.xml`, the definitions defined in `zscript-config` are applied to all Web applications sharing the same `zk.jar`.

In other words, the scripting language defined here are visible to all Web applications sharing the same `zk.jar`. Furthermore, it may cause errors in another Web application, if the classes or resources are available only in the Web application defining this.

Thus, if it is an issue, just put `zk.jar` and relevant ZK libraries under the `WEB-INF/lib` directory.

The language-name Element

[Required]

It specifies the language name. It is case insensitive. The previous implementation with the same language name will be replaced if any.

The interpreter-class Element

[Required]

It specifies the implementation class. It must implement the `org.zkoss.zk.scripting.Interpreter` interface. Instead of implementing it from scratch, you can derive from the `org.zkoss.zk.scripting.util.GenericInterpreter` class. If you want to support the hierarchical scopes (i.e., one interpreter scope per namespace), it can also implement the `org.zkoss.zk.scripting.HierachicalAware` interface.

The device-config Element

It specifies a device. A device represents a client. Different clients have different implementation. Currently there are two types: `ajax` and `mil`. They represents the Web browsers with Ajax, and the mobile device with Mobile Interactive Language¹⁰. It is used to create an instance returned by the `getDevice` method of the `Desktop` interface.

The allowed child element is `device-type`, `device-class`, `timeout-uri`, and `unavailable-message`. You might have multiple `device-config` elements in one `zk.xml`.

```
<device-config>
  <device-type>ajax</device-type>
  <device-class>my.MyAjaxDevice</device-class>
  <timeout-uri>/my-timeout.zul</timeout-uri>
  <unavailable-message><![CDATA[
<p style="color:red">Sorry, JavaScript must be enabled in order for you to use
KillApp.</p>
]]></unavailable-message>
</device-config>
```

Note: Unlike most other configurations defined in `WEB-INF/zk.xml`, the definitions defined in `device-config` are applied to all Web applications sharing the same `zk.jar`. Refer to the `zscript-config` element for more information.

The device-type Element

[Required]

It specifies the device type. The previous implementation with the same device type will be replaced if any.

The device-class Element

[Optional]

It specifies the implementation class. The class must implement the `org.zkoss.zk.device.Device` interface. Instead of implementing it from scratch, you can derive from the proper implementation, such as `AjaxDevice` and `MilDevice`.

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.

¹⁰ MIL is a ZK markup language used to communicate with the mobile devices.


```
<timeout-uri></timeout-uri>
```

The unavailable-message Element

[Optional]

It specifies the message that will be displayed if the client doesn't support this device.

The error-page Element

```
<error-page>
  <device-type>[ajax|mil]</device-type>
  <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 examining the exception type one-by-one. If none is found, it shows, by default, an alert message at the client.

The `device-type` element is optional. If omitted, `ajax` is assumed. If you want to specify an error page for mobile devices, it has to be `mil`.

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) {
    ...
}
```