

ZUML Reference

For ZK 5.0.9

Contents

Articles

ZUML Reference	1
ZUML	1
Languages	2
ZUL	6
XHTML	6
XML	7
Namespaces	8
Client	9
Client Attribute	10
Native	11
ZK	12
Elements	13
attribute	13
custom-attributes	15
variables	16
zk	18
zscript	21
Attributes	23
apply	24
forEach	25
forEachBegin	26
forEachEnd	26
forward	27
fulfill	28
if	29
unless	29
use	30
Texts	30
Processing Instructions	32
component	33
evaluator	36
forward	39
function-mapper	40
header	41

import	42
init	44
link	45
meta	46
page	47
root-attributes	51
script	52
style	53
taglib	54
Custom Taglib	55
variable-resolver	57
xel-method	58
EL Expressions	59
Literals	59
Operators	60
Type Coercion	61
Implicit Objects	62
applicationScope	62
arg	63
componentScope	64
cookie	64
desktop	65
desktopScope	65
each	66
event	67
execution	68
header	68
headerValues	68
forEachStatus	69
labels	70
page	71
pageContext	71
pageScope	72
param	72
paramValues	73
requestScope	73
self	74
session	74

sessionScope	75
spaceOwner	75
spaceScope	76
Core Methods	76
boolean	77
browser	77
cat	78
cat3	78
cat4	79
cat5	79
char	80
class	80
decimal	81
encodeURIComponent	81
endsWith	82
escapeXML	83
getCurrentLocale	83
indexOf	84
int	84
isInstance	85
join	85
l	86
l2	87
lastIndexOf	88
length	88
number	89
property	89
new	90
new1	90
new2	91
new3	91
split	92
startsWith	92
string	93
substring	93
testCurrentLocale	94
toLowerCase	94
toUpperCase	95

trim	95
Extensions	96
zscript	96
EL Expressions	98

References

Article Sources and Contributors	99
----------------------------------	----

ZUML Reference

Documentation:Books/ZUML_Reference

If you have any feedback regarding this book, please leave it here.

<comment>http://books.zkoss.org/wiki/ZUML_Reference</comment>

ZUML

ZUML (ZK User Interface Markup Language) is based on XML. Similar to HTML and XUL, it is used to describe UI in an easy-to-understand format.

In a ZUML document, each XML element instructs the ZK Loader which component to create. Each XML attribute describes what value to be assigned to the created component. Each XML processing instruction describes how to process the whole page, such as the page title. For example,

```
<?page title="Super Application"?>
<window title="Super Hello" border="normal">
  <button label="hi" onClick='alert("hi")' />
```

where the first line specifies the page title, the second line creates a root component with title and border, and the third line creates a button with label and an event listener.

For introduction of ZUML, please refer to ZK Developer's Reference. If you are not familiar with XML, please take a look at XML Background first.

Languages

Overview

A language (`LanguageDefinition` ^[1]) is a collection of component definitions. It is also known as a component set.

For example, `Window` ^[2], `Button` ^[3] and `Combobox` ^[4] all belong to the same language called `xul/html`. It is a ZK variant of XUL (and also known as `zul`).

Component designers are free to designate a component definition to any component set they prefer, as long as there is no name conflict^[5].

For introduction of languages vs standard namespaces, please refer to ZK Developer's References.

Language Identification

When parsing a ZUML document, ZK Loader has to decide the language that a XML element is associated, such that the correct component definition (`ComponentDefinition` ^[6]) can be resolved. For example, in the following example, ZK needs to know if `window` belongs to the `xul/html` language, so its component definition can be retrieved correctly.

```
<window>
```

ZK Loader takes the following steps to decide the language an XML element is associated with:

1. It assumes a default language for a ZUML document. The default language is decided by the filename's extension (see below).
2. If an XML element has no namespace prefix, then
 1. Handle it specially, if the element is a special ZK element, such as `zk` and `attribute`.
 2. Look up the component definition belonging to the default language, otherwise.
3. If an XML element has a prefix, then the XML namespace is used to resolve:
 1. Handle it specially, if the XML namespace is one of the standard namespaces, such as `native` and `client`.
 2. Look up the language with the given XML namespace, otherwise
 3. Then, look up the component definition from the language found in the previous step

Filename Extension

The default language is decided based on the extension of the filename (`LanguageDefinition.getByExtension(java.lang.String)` ^[7]). In addition, a language is associated with one or multiple extensions (defined by the component developer). For example, the extensions associated with the `xul/html` language are `zul` and `xul`, while the `xhtml` language (aka., a component set) is associated with the extensions including `zhtml`, `html`, `html`, and `xhtml`.

Thus, if a file or URI whose extension is `zul` or `xul`, the default language will be assumed to be the `xul/html` language.

Filename Extension vs URL Mapping

The association of extensions with a language is defined in a language. However, to really have ZK Loader to process a particular file, you have to configure WEB-INF/web.xml correctly. For example, if you want to map all *.xul files to ZK Loader, you could add the following to WEB-INF/web.xml:

```
<servlet-mapping>
  <servlet-name>zKLoader</servlet-name>
  <url-pattern>*.xul</url-pattern>
</servlet-mapping>
```

If the extension of the mapped URL does not match any language, the xul/html language is assumed.

XML Namespace

In addition to extension association, a language is also associated with a unique XML namespace. Thus, you can identify the language for a given XML element by the use of XML namespace.

With the XML namespace, you could:

1. Map a default language for a unknown extension
2. Mix two or more languages in one ZUML document

Map a default language for a unknown extension

For example, you map ZK Loader to *.foo, which is not associated with any language. Then, you have to specify the XML namespace as shown in the following example:

```
<window xmlns="http://www.zkoss.org/2005/zul">
  ...
```

where the xmlns attribute declares a XML namespace to associate all element without explicit prefix, such as window in this case. Furthermore, http://www.zkoss.org/2005/zul is the unique XML namespace associated with the xul/html namespace.

Mix two or more languages in a ZUML document

If you want to use several languages in the same XML document, you could use XML namespaces to distinguish them too. For example, the xhtml language's namespace is http://www.w3.org/1999/xhtml, and we could mix the use as follows.

```
<window xmlns:h="http://www.w3.org/1999/xhtml">
  <h:table>
    <h:tr>
      <h:td>
        <button/>
      </h:td>
    </h:tr>
  </h:table>
</window>
```

Notice that, when using the xhtml language, table, tr and td are also components though they are very simple -- a simple wrapper of HTML tags. However, there is a better way to generate HTML tags: the native namespace. It generates HTML tags directly without maintaining the component^[8]. The associated XML namespace of the native

namespace is <http://www.zkoss.org/2005/zk/native>, so we can rewrite the previous example to be more efficient:

```
<window xmlns:h="http://www.zkoss.org/2005/zk/native">
  <h:table>
    <h:tr>
      <h:td>
        <button/>
      </h:td>
    </h:tr>
  </h:table>
</window>
```

[1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/metainfo/LanguageDefinition.html#>

[2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Window.html#>

[3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Button.html#>

[4] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Combobox.html#>

[5] For more information please refer to ZK Component Development Essentials

[6] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/metainfo/ComponentDefinition.html#>

[7] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/metainfo/LanguageDefinition.html#getByExtension\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/metainfo/LanguageDefinition.html#getByExtension(java.lang.String))

[8] For more information please refer to the Native Namespace section

XML Namespace with Shortcut

To make it easy to specify a namespace, you could specify a shortcut instead of the full namespace URI. For languages, the shortcut is the last word of the namespace URI. For example, zul for <http://www.zkoss.org/2005/zul>, and xhtml for <http://www.w3.org/1999/xhtml>. Thus, we can simply the previous example as follows.

```
<window xmlns:h="xhtml">
  <h:table>
    <h:tr>
      <h:td>
        <button/>
      </h:td>
    </h:tr>
  </h:table>
</window>
```

Standard Languages

ZK provides three different languages (aka., component sets): xul/xhtml, xhtml and xml. The xul/xhtml and xhtml languages can be used for any modern browser (Ajax assumed), while the zml language is used for generating XML document (non-Ajax). The developers are free to add their own language^[1].

[1] Notice that there are so-called Standard Namespaces associated with XML namespaces (for a ZUML document) to provide special functionality (than specify components).

Language	Description
xul/html	<p>Name: xul/html (aka., zul) File Extensions: zul, xul Namespace: http://www.zkoss.org/2005/zul Namespace shortcut: zul Device: Ajax</p> <p>XUL-compliant component sets. We adopt XUL (https://developer.mozilla.org/En/XUL) for this language, if the specification is applicable. For more information, please refer to ZK Component Reference.</p>
xhtml	<p>Name: xhtml File Extensions: zhtml, xhtml, html, htm Namespace: http://www.w3.org/1999/xhtml Namespace shortcut: xhtml Device: Ajax</p> <p>XHTML-compliant component sets. It is one-to-one mapping of XHTML tags to ZK components. Since they are components, you can add and remove them dynamically (and control it at the server). For more information please refer to the XHTML Namespace section or ZK Component Reference.</p> <p>Performance Tip: The XHTML language is designed to allow application to modify the client dynamically (at the server). If you don't need it (it is generally true), you should use the Native namespace instead. For more information, please refer to Performance Tips.</p>
xml	<p>Name: xml File Extensions: xml Namespace: http://www.zkoss.org/2007/xml Namespace shortcut: xml Device: XML Available only ZK EE</p> <p>XML component sets. It is used to generate (static) XML document. For more information please refer to the XML section.</p>

Version History

Version	Date	Content
5.0.4	August, 2010	The shortcut was introduced to make it easy to specify a standard namespace, such as native, client and zk.
5.0.5	October, 2010	The shortcut was introduced to make it easy to specify a component set, such as zul and zhtml.

ZUL

```
Name: xul/html (aka., zul)
File Extensions: zul, xul
Namespace: http://www.zkoss.org/2005/zul
Namespace shortcut: zul
Device: Ajax
```

XUL-compliant component sets. We adopt XUL ^[1] for this language, if the specification is applicable, such as Tabbox ^[2] and Grid ^[3]. It basically contains all rich components for the Ajax devices (i.e., the browsers).

For more information, please refer to ZK Component Reference.

Version History

Version	Date	Content
---------	------	---------

References

- [1] <https://developer.mozilla.org/En/XUL>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Tabbox.html#>
- [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Grid.html#>

XHTML

```
Name: xhtml
File Extensions: zhtml, xhtml, html, htm
Namespace: http://www.w3.org/1999/xhtml
Namespace shortcut: xhtml
Device: Ajax
```

XHTML-compliant component sets. It is one-to-one mapping of XHTML tags to ZK components. Since they are components, you can add and remove them dynamically (and control it on the server). For more information please refer to the XHTML Namespace section or ZK Component Reference.

Performance Tip: The XHTML language is designed to allow applications to modify the client dynamically (at the server). If you don't need it (it is generally true), you should use the Native namespace instead. For more information, please refer to HTML Tags and Performance Tips.

Version History

Version	Date	Content
---------	------	---------

XML

```
Name: xml
File Extensions: xml
Namespace: http://www.zkoss.org/2007/xml
Namespace shortcut: xml
Device: XML
Available only ZK EE
```

XML component sets. It is used to generate (static) XML document, such as RSS feed ^[1]. For introduction please refer to ZK Developer's Reference.

Most of XML elements with the XML namespace are mapped to a general XML component (XmlNativeComponent ^[2]) that will generate the element and all its attributes to the client directly.

However, the XML component set also provide some components for different functionality. For more information please refer to ZK Component Reference.

Version History

Version	Date	Content
---------	------	---------

References

[1] <http://www.whatisrss.com/>

[2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zml/XmlNativeComponent.html#>

Namespaces

Standard Namespaces

Standard namespaces are not languages. That means they are *not* used to provide component definitions. Rather, they are used to provide special functionality to ZUML.

For introduction of languages vs standard namespaces, please refer to ZK Developer's References.

Namespace	Description
zk	<p>Name: zk Namespace: <code>http://www.zkoss.org/2005/zk</code> Namespace shortcut: zk</p> <p>It is the reserved namespace for specifying ZK specific elements and attributes, such as the zk element and the unless attribute. For more information please refer to the ZK Namespace section.</p>
native	<p>Name: native Namespace: [1] Namespace shortcut: native</p> <p>It is the reserved namespace for specifying native elements. A native element represents a native tag at the client. For browsers, a native element represents a HTML tag. Unlike the xhtml language, there is no component associated with, so the performance is much better but you cannot change it dynamically.</p> <p>For more information please refer to the Native Namespace section.</p>
client	<p>Name: client Namespace: [2] Namespace shortcut: client</p> <p>It is the reserved namespace for specifying client-side event listener and overrides. For more information please refer to the Client Namespace section.</p>
client attribute	<p>Name: client attribute Namespace: [3] Namespace shortcut: client/attribute</p> <p>It is the reserved namespace for specifying client-side DOM attributes. Unlike the client namespace, which assigns something to widgets, the client/attribute namespace assigns to the DOM tree directly.</p> <p>For more information please refer to the Client Attribute Namespace section.</p>
xhtml	<p>Name: xhtml Namespace: <code>http://www.w3.org/1999/xhtml</code> Namespace shortcut: xhtml</p> <p>For more information please refer to the the Languages section.</p>
zul	<p>Name: xul/html Namespace: <code>http://www.zkoss.org/2005/zul</code> Namespace shortcut: zul</p> <p>For more information please refer to the the Languages section.</p>
xml	<p>Name: xml Namespace: <code>http://www.zkoss.org/2007/xml</code> Namespace shortcut: xml</p> <p>For more information please refer to the the Languages section.</p>

For more information of XHTML, ZUL and other component sets, please refer to the Languages section.

Version History

Version	Date	Content
---------	------	---------

References

- [1] <http://www.zkoss.org/2005/zk/native>
- [2] <http://www.zkoss.org/2005/zk/client>
- [3] <http://www.zkoss.org/2005/zk/client/attribute>

Client

```
Name: client
Namespace: http://www.zkoss.org/2005/zk/client
Namespace shortcut: client
```

It is the reserved namespace for specifying client-side event listener and overrides.

For example,

```
<combobox xmlns:w="client" w:onFocus="this.open()" />
```

For more information, please refer to ZK Client-side Reference.

Version History

Version	Date	Content
---------	------	---------

Client Attribute

```
Name: client attribute
Namespace: http://www.zkoss.org/2005/zk/client/attribute
Namespace shortcut: client/attribute
```

It is the reserved namespace for specifying client-side DOM attributes. Unlike the client namespace, which assigns something to widgets, the client/attribute namespace assigns additional DOM attributes to the DOM tree directly at the client.

Notice that if the widget's DOM output (`Widget.redraw(_global_.Array)` ^[1]) also has the same DOM attribute, both of them will be generated and it is technically not legal. Thus, you shall prevent the DOM attributes that widget might output.

For example, suppose you want to listen to the onload event, and then you can do as follows ^[2].

```
<iframe src="http://www.google.com" width="100%" height="300px"
  xmlns:ca="client/attribute"
  ca:onload="do_whater_you_want ()" />
```

If the attribute contains colon or other special characters, you can use the attribute element as follows.

```
<div xmlns:ca="client/attribute">
  <attribute ca:name="ns:whatever">
    whatever_value_you_want
  </attribute>
</div>
```

The other use of the client-attribute namespace is to specify attributes that are available only to certain browsers, such as accessibility and Section 508 ^[3].

[1] [http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#redraw\(_global_.Array\)](http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#redraw(_global_.Array))

[2] For more information, please refer to ZK Component Reference: iframe.

[3] <http://www.section508.gov/index.cfm?FuseAction=Content&ID=12#Web>

Version History

Version	Date	Content
5.0.3	July 2010	The client-attribute namespace was introduced.

Native

```
Name: native
Namespace: http://www.zkoss.org/2005/zk/native
Namespace shortcut: native
```

It is the reserved namespace for specifying native elements. A native element represents a native tag at the client. For browsers, a native element represents a HTML tag. Unlike the xhtml language, there is no component associated with, so the performance is much better but you cannot change it dynamically.

```
<n:table xmlns:n="native">
  <n:tr>
    <n:td>Username</n:td>
    <n:td><textbox/></n:td>
  </n:tr>
  <n:tr>
    <n:td>Password</n:td>
    <n:td><textbox type="password"/></n:td>
  </n:tr>
</n:table>
```

where `n:table`, `n:tr` and `n:td` are native, i.e., they are generated directly to the client without creating a component for each of them.

Notice that ZK Loader assumes any element name with the native namespace is correct and generated to the client directly. It is your job to make sure there is no typo or other errors.

Version History

Version	Date	Content
---------	------	---------

ZK

Name: zk
Namespace: http://www.zkoss.org/2005/zk
Namespace shortcut: zk

It is the standard namespace for specifying ZK specific elements and attributes, such as the zk element and unless attribute.

By default, ZK Loader will detect if a XML element or attribute is a special element or attribute, and then handle it differently. However, if the default XML namespace is the native namespace or a component set that accepts any element name, such as the XHTML language, you have to specify the zk namespace. Otherwise, they will be interpreted as a component. For example,

```
<html xmlns="native" xmlns:u="zul" xmlns:zk="zk">
  <head>
    <title>ZHTML Demo</title>
  </head>
  <body>
    <script type="text/javascript">
      function woo() { //running at the browser
      }
    </script>
    <zk:zscript>
      void addItem() { //running at the server
      }
    </zk:zscript>
    <u:window title="HTML App">
      <input type="button" value="Add Item"
        onClick="woo()" zk:onClick="addItem()" />
    </u:window>
  </body>
</html>
```

Version History

Version	Date	Content
---------	------	---------

Elements

Each XML element represents a component, except special elements like `<zk>` and `<attribute>`.

In the following sections, we will discuss the special elements one-by-one.

attribute

Syntax:

```
<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>
```

In addition, you can specify a XML fragment as the value of the attribute. The XML fragment is so-called the native content.

```
<html>
  <attribute name="content">
    <ol>
      <li forEach="apple, orange">${each}</li>
    </ol>
  </attribute>
</html>
```

where `ol` and `li` are part of the native content. They are not ZK components. They will be eventually converted to a String instance and assigned to the specified attribute. If values has three elements, the above example is equivalent to the following:

```
<html>
  <attribute name="content">
    <ol>
      <li>apple, orange</li>
      <li>orange</li>
    </ol>
  </attribute>
```

```
</html>
```

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.

Version History

Version	Date	Content
---------	------	---------

custom-attributes

Syntax:

```
<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 or page depending on this parent]

Specifies the scope to which the custom attributes are associated. If not specified and enclosed with a component, the component is the default scope. If not specified and not enclosed with a component, the default scope is page (since 5.0.8). For example,

```
<zkc>
  <custom-attributes a="A"/><!-- assign to page's attribute (since 5.0.8) -->
  <button label="show a" onClick='alert(page.getAttribute("a"))'/>
  <button label="show b" onClick='alert(self.getAttribute("b"))'>
    <custom-attributes b="B"/> <!-- assign to the button's attribute -->
  </button>
</zkc>
```

composite

[Optional] [Default: none]

Specifies the format of the value. It could be `none`, `list` or `map`.

By default, the value is assigned to the attribute directly after evaluating EL expressions, if any. For example, "apple, \${more}" is evaluated to "apple, orange", if more is "orange", and assigned to the attribute.

If you want to specify a list of values, you can specify the `composite` attribute with `list` as follows.

```
<custom-attributes simple="apple, ${more}" composite="list"/>
```

Then, it is converted to a list with two elements. The first element is "apple" and the second "orange".

If you want to specify a map of values, you can specify the `composite` attribute with `map` as follows.

```
<custom-attributes simple="juice=apple, flavor=${more}" composite="map"/>
```

Then, it is converted to a map with two entries. The first entry is ("juice", "apple") and the second ("flavor", "orange").

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.

Version History

Version	Date	Content
5.0.8	July, 2011	The custom-attributes element is allowed to be placed under the page definition directly.

variables

Syntax:

```
<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.

composite

[Optional] [Default: none]

Specifies the format of the value. It could be `none`, `list` or `map`.

By default, the value is assigned to the variable directly after evaluating EL expressions, if any. For example, "apple, \${more}" is evaluated to "apple, orange", if more is "orange", and assigned to the variable.

If you want to specify a list of values, you can specify the `composite` attribute with `list` as follows.

```
<variables simple="apple, ${more}" composite="list"/>
```

Then, it is converted to a list with two elements. The first element is "apple" and the second "orange".

If you want to specify a map of values, you can specify the `composite` attribute with `map` as follows.

```
<variables simple="juice=apple, flavor=${more}" composite="map"/>
```

Then, it is converted to a map with two entries. The first entry is ("juice", "apple") and the second ("flavor", "orange").

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.

Version History

Version	Date	Content
---------	------	---------

zk

Syntax:

```
<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 should 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 should 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.

switch

[Optional] [Default: none]

Provide the context for mutually exclusive evaluation. The value specified in this attribute is called the switch condition.

```
<zk switch="{condition}"/>
```

The only allowed children are the `zk` elements.

For more examples, please refer to ZK Developer's Reference: Conditional Evaluation.

case

[Optional] [Default: none]

Provides an alternative within the switch evaluation.

```
<zk case="apple"/>
```

If the value is a string starting and ending with slash, such as `/a[p]*/`, it is considered as a regular expression, which is used to match the switch condition.

```
<zk case="/a[a-z]*/"/>
```

You can specify multiple cases by separating them with comma.

```
<zk case="apple, ${special}"/>
```

For more examples, please refer to ZK Developer's Reference: Conditional Evaluation.

choose

[Optional] [Default: none]

Provide the context for mutually exclusive evaluation.

```
<zk choose="">
```

As shown, the value of the `choose` attribute is always empty, since it is used only to identify the range of mutually exclusive conditional evaluation.

The only allowed children are the `zk` elements.

For more examples, please refer to ZK Developer's Reference: Conditional Evaluation.

when

[Optional] [Default: none]

Provides an alternative within the choose evaluation.

```
<zk when="{fruit == 'apple'}">
```

It is evaluated if the condition matches.

For more examples, please refer to ZK Developer's Reference: Conditional Evaluation.

Version History

Version	Date	Content
---------	------	---------

zscript

Syntax:

```
<zscript [language="''Java''|JavaScript|Ruby|Groovy"]>Scripting codes</zscript>
<zscript src="''a_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 with the use of `language` attribute^[1].

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"/>
```

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

For introductory of `zscript`, please refer to ZK Developer's Reference.

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

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 should 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 the Web applications from peeking other's content^[1].
3. It accepts "*" for loading browser and Locale dependent style sheet.

The algorithm to resolve "*" is as follows.

- If there is one "*" 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^[2]. If the last "*" will be replaced with a proper

Locale as described above.

- All other "*" are ignored.

Notes

[1] Refer to the `getContext` meth from the `javax.servlet.ServletContext` interface.

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

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.

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.

For instance, in the following example, the interpreter is loaded and the `zscript` element is evaluated, only when the button is clicked:

```
<syntax lang="xml" > <window id="w">
```

```
  <zscript deferred="true">
    void addMore() {
      new Label("More").setParent(w);
    }
  </zscript>
  <button label="Add" onClick="addMore()" />
```

```
</window> </syntax>
```

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.

Version History

Version	Date	Content
---------	------	---------

Attributes

Each attribute, except special attributes like `if` and `forEach`, represents a value that should be assigned to a property of a component after it is created. For example, when an attribute, say, `foo</foo>`, is specified, ZK Loader will assume there is a method called `setFoo` that accepts a single argument. If there are multiple methods with the same name, ZK Loader will use the one that matches the argument most (in term of the argument's class).

For example, suppose `${foo}` is evaluated to an integer in the following example, ZK Loader will invoke `Window.setMode(int)` ^[1], rather than `Window.setMode(java.lang.String)` ^[2].

```
<window mode="${foo}">
...

```

The values of attributes usually consist of EL expressions. For example,

```
<listbox forEach="${matrix}">
  <listitem label="${forEachStatus.previous.each.label}: ${each}" forEach=${each.items}
</listbox>

```

There are several ways to associate Java objects with EL expressions^[3].

1. Implement a variable resolver (`VariableResolver` ^[4]) and specify it with the `variable-resolver` directive.
2. Return the object in a static method and specify it in `xel-method`
3. Declare multiple static methods in a taglib and declare it in taglib
4. Construct them in `zscript`

In the following sections, we will discuss the special attributes one-by-one.

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Window.html#setMode\(int\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Window.html#setMode(int))

[2] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Window.html#setMode\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Window.html#setMode(java.lang.String))

[3] For introductory, please refer to ZK Developer's Reference.

[4] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/VariableResolver.html#>

apply

Syntax

```
apply="a-class-name"  
apply="class1, class2, ..."  
apply="${EL_returns_a_class_or_a_collection_of_classes}"  
apply="${EL_returns_an_instance_or_a_collection_of_Composer_instances}"
```

It specifies a class, a collection of classes that are used to initialize the component. The class must implement the `Composer` ^[1] interface. And then, you can do the initialization in the `doAfterCompose` method, since it is called after the component and all its children are instantiated.

```
<window apply="foo.MyComposer" />
```

In addition, you specify a `Composer` instance, or a collection of `Composer` instances by the use of EL expressions.

Note: the EL expressions are, if specified, evaluated before the component is instantiated. So you cannot reference to the component. Moreover, the `self` variable references to the parent component, if any, or the current page, if it is the root component, in the EL expressions specified in this attribute.

If you want more control such as handling the exception, you can also implement the `ComposerExt` ^[2] interface.

If you have a composer that you'd like to apply to every page, you don't need to specify it in every page. Rather, you could register a system-level composer. For more information, please refer to ZK Developer's Reference: System-level Composers.

Version History

Version	Date	Content
---------	------	---------

References

[1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Composer.html#>

[2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ComposerExt.html#>

forEach

Syntax:

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

The `forEach` attribute is used to specify a collection of object such that the XML element it belongs will be evaluated repeatedly for each object of the collection.

There are two formats. First, you specify a value without comma. The value is usually a collection of objects, such that the associated element will be evaluated repeatedly against each object in the collection. If this attribute is not specified or empty, it will be ignored. If non-collection object is specified, it is evaluated only once as if a single-element collection is specified.

Second, you can specify a list of values by separating them with commas. Then, the associated element will be evaluated repeatedly against each value in the list.

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>
```

When ZK Loader iterates through items of the give collection, it will update two implicit objects: `each` and `forEachStatus`. The `each` variable represents the item being iterated, while `forEachStatus` is an instance of `ForEachStatus`^[1], from which you could retrieve the index and the previous `forEach`, if any.

If you prefer to iterate only a portion of a collection, you could specify `forEachBegin` and/or `forEachEnd`.

Fore more examples, please refer to ZK Developer's Reference: Iterative Evaluation.

Version History

Version	Date	Content
---------	------	---------

References

[1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ForEachStatus.html#>

forEachBegin

Syntax:

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

It is used with the `forEach` attribute to specify the index (starting from 0) that the iteration should 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.

Version History

Version	Date	Content
---------	------	---------

forEachEnd

Syntax:

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

The `forEach` attribute is used to specify the index (starting from 0) which element iteration should end 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.

Version History

Version	Date	Content
---------	------	---------

forward

Syntax:

```
forward="originalEvent=targetId1/targetId2.targetEvent"
forward="originalEvent=targetId1/targetId2.targetEvent(eventData)"
forward="originalEvent=${el-target}.targetEvent(${el-eventdata})"
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 forward event is an instance of the `ForwardEvent` ^[1] class. you can invoke `ForwardEvent.getOrigin()` ^[2]. to retrieve the original event.

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.

You could specify any application-specific data in the forward condition by surrounding it with the parenthesis as shown below.

```
<button forward="onCancel(abort)" /><!-- "abort" is passed -->
<button forward="onPrint(${inf})" /><!-- the object returned by ${inf} is passed -->
```

Then, the application-specific data can be retrieved by the use of `ForwardEvent.getData()` ^[3].

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

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

The target component and the event data can be specified in EL expressions, while the event names cannot.

The target component can also be specified using component Path ^[4] within ZUML page. This is especially useful if target component is in different IdSpace ^[5]

```
<button forward="//mainPage/mainWindow.onSave" /> <!-- default forward event is onClick -->
```

Version History

Version	Date	Content
---------	------	---------

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/ForwardEvent.html#>
- [2] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/ForwardEvent.html#getOrigin\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/ForwardEvent.html#getOrigin())
- [3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/ForwardEvent.html#getData\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/ForwardEvent.html#getData())
- [4] http://books.zkoss.org/wiki/ZK_Developer's_Guide/ZK_in_Depth/Component_Path_and_Accessibility/Access_UI_Component
- [5] http://books.zkoss.org/wiki/ZK_Developer's_Reference/UI_Composing/ID_Space

fulfill

Syntax:

```
fulfill="condition"
fulfill="condition_1, condition_2, ..."
fulfill="condition=a_uri"
fulfill="condition_1, condition_2=a_uri, ..."
```

and the fulfill condition (`condition`, `condition_1` and `condition_2`) could be one of the following:

- *event-name*
- *target-id.event-name*
- *id1/id2/id3.event-name*
- *\${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.

If URI (`a_uri`) is specified, the ZUML document of the given URI will be loaded and rendered as children of the associated component. Notice that you could specify at most one URI in a `fulfill` attribute.

For more information, please refer to ZK Developer's Reference: On-demand Evaluation.

The onFulfill Event

After ZK applies the fulfill condition, i.e., creates all descendant components, it fires the `onFulfill` event with an instance of `FulfillEvent` ^[1] to notify the component for further processing if any.

For example, if you use the `wireVariables` method of the `Components` ^[2] class, you might have to call `wireVariables` again to wire the new components in the `onFulfill` event.

```
<div fulfill="b1.onClick, b2.onOpen" onFulfill="Components.wireVariables(self, controller
    ...
</div>
```

Version History

Version	Date	Content
---------	------	---------

References

[1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/FulfillEvent.html#>

[2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Components.html#>

if

Syntax:

```
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.

For example, suppose you want to place either one label or another in a column of a grid, you might use something like this:

```
<row forEach="{salesPersonList}">
  <label value="{each.salesPersonName}" if="{each.hasSalesPerson=='Y'}"/>
  <label value="-" style="float:center" if="{each.hasSalesPerson!='Y'}"/>
</row>
```

Version History

Version	Date	Content
---------	------	---------

unless

Syntax:

```
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.

Version History

Version	Date	Content
---------	------	---------

use

Syntax:

```
use="a-class-name"
use="${EL_returns_a_class_or_a_class_name}"
use="${a_component}"
```

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, `Window` ^[2].

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

If an EL expression is used, it can return a class name, a class instance, or a component instance. Notice that, if the expression returns a component, the component should not belong to any pages.

Version History

Version	Date	Content
---------	------	---------

Texts

In general, a XML text is interpreted as a label component. For example,

```
<window>
  Begin ${foo.whatever}
</window>
```

is equivalent to

```
<window>
  <label value="Begin ${foo.whatever}"/>
</window>
```

Components consider the nested content as property

However, a component can be designed to accept the nested text as the value of a component property. In other words, a component designer could decide to make ZK Loader interpret the nested text as the value of a predefined property. For example, `Html` ^[1] is one of this kind of components, and

```
<html>Begin ${foo.whatever}</html>
```

is equivalent to

```
<html content="Begin ${foo.whatever}"/>
```

It is designed to make it easy to specify multiple-line value, so it is usually used by particular components that requires the multi-line value.

Here is a list of components that interprets the XML text as a property's value.

Component Name	Property Name	Method
a	label	A.setLabel(java.lang.String) ^[2]
button	label	Button.setLabel(java.lang.String) ^[3]
comboitem	content	Comboitem.setContent(java.lang.String) ^[4]
html	content	Html.setContent(java.lang.String) ^[5]
label	value	Label.setValue(java.lang.String) ^[6]
script	content	Script.setContent(java.lang.String) ^[7]
style	content	Style.setContent(java.lang.String) ^[8]
tab	label	Tab.setLabel(java.lang.String) ^[9] (since 5.0.7)

The nested XML content

[since 6.0.0]

Since ZK 6, components that consider the text as a property's value will accept the XML fragment. For example,

```
<html>
  <ol style="border: 1px solid blue">
    <li>Apple</li>
    <li>Orange</li>
  </ol>
</html>
```

In other words, you don't have to escape the special characters (< and >) with CDATA. In addition, you could leverage the full power of ZUML such as the zk element and the forEach attribute. For example,

```
<html>
  <ol>
    <li forEach="Apple, Orange">${each}</li>
  </ol>
</html>
```

Note that the nested content is part of the ZUML page, so it must be a legal XML document.

Version History

Version	Date	Content
5.0.7	April 2011	Tab ^[10] allow the XML text as the label.
6.0.0	September 2011	The nested XML content was supported.

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Html.html#>
- [2] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/A.html#setLabel\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/A.html#setLabel(java.lang.String))
- [3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Button.html#setLabel\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Button.html#setLabel(java.lang.String))
- [4] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Comboitem.html#setContent\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Comboitem.html#setContent(java.lang.String))
- [5] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Html.html#setContent\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Html.html#setContent(java.lang.String))
- [6] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Label.html#setValue\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Label.html#setValue(java.lang.String))
- [7] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Script.html#setContent\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Script.html#setContent(java.lang.String))
- [8] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Style.html#setContent\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Style.html#setContent(java.lang.String))
- [9] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Tab.html#setLabel\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Tab.html#setLabel(java.lang.String))
- [10] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Tab.html#>

Processing Instructions

Each XML processing instruction specifies the instruction how to process the XML document. It is called directives in ZK. For example, the following specifies the page title and style.

```
<?page title="Grey background" style="background: grey"?>
```

Notice that there should be *no* whitespace between the question mark and the processing instruction's name (i.e., page in the above example).

component

Syntax:

```
<?component name="myName" macroURI="/mypath/my.zul" [inline="true|false"]
  [apply="composer"] [prop1="value1"] [prop2="value2"]... ?>
<?component name="myName" [class="myPackage.myClass"]
  [extends="nameOfExistComponent"]
  [moldName="myMoldName"] [moldURI="/myMoldURI"]
  [apply="composer"] [prop1="value1"] [prop2="value2"]... ?>
```

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

The by-macro Format

Syntax:

```
<?component name="myName" macroURI="/mypath/my.zul"
  [apply="composer"] [language="xul/html"] [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 `macroURI` 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 (through 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

Syntax:

```
<?component name="myName" [class="myPackage.myClass"]
  [extends="nameOfExistComponent"]
  [moldName="myMoldName"] [moldURI="/myMoldURI"]
  [apply="composer"] [language="xul/html"] [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 `Component`^[1] 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 one `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 the `extends` element with the component's name to extend from (aka., *extende*). In other words, if `extends` is specified, the definition of the *extende* is loaded as the default value and then override only properties that are

specified in this directive.

If the name of extender and extender are the same, it means the extender will override the definition of extender.

For example, assume you want to use `MyWindow` instead of the default `window`, `Window`^[2] for all windows defined in this ZUML page. Then, you can declare it as follows.

```
<?component name="window" extends="window" 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" extends="button" 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" extends="button" 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 initial properties, you could invoke the `applyProperties` method as follows.

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

Attributes

apply

[Optional]

[Since 3.6.0]

The `apply` condition, which is a list of composer's class names or EL expressions. If an EL expression is specified, it must return either a class instance, a class name, a composer instance or null.

Notice that the list of composers specified here is always applied even if the component has its own `apply` condition. For example, both `BaseComposer` and `FooComposer` are applied in the following example,

```
<?component name="window" extends="window" apply="BaseComposer"?>
<window apply="FooComposer">
</window>
```

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`.

For implementing a macro component, please refer to ZK Developer's Reference.

extends

[Optional]

Specifies the component name to extend from. The existent definition of the specified name will be loaded to initialize the new component definition. In other words, it *extends* the existent definition instead of defining a brand-new one.

language

[Optional] [Since ZK 5.0.0]

Specifies which language to look for the component definition to extends from. If omitted, the page's language is assumed.

Notice that the new defined component is visible only to the associate page. The language attribute is used for locating the component definition specified in the extends attribute. For example, the following statement works even if it is used in a ZHTML file.

```
<?component name="foo" extends="button" language="xul/html"?>
```

macroURI

[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.

moldName

[Optional] [Default: default]

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

moldURI

[Optional] [EL is allowed]

```
moldURI="~/zul/in-my-jar.dsp"  
moldURI="/WEB-INF/in-my-web.dsp"  
moldURI="/jsp-or-other-servlet"  
moldURI="class:com.mycompany.myrender"
```

Used with the by-class format to specify the mold URI. If `moldURI` is specified but `moldName` 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.

Version History

Version	Date	Content
---------	------	---------

References

[1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#>

evaluator**Syntax:**

```
<?evaluator [name="..."] [class="..."] [import="..."]?>
```

It specifies how to evaluate XEL expressions.

name[Optional] [Default: *none*] [Case insensitive]

The name of the implementation used to evaluate the XEL expressions. There are two ways to specify the implementation. One is the name attribute. The other is the class attribute.

For example, if you want to use MVEL^[1], you can specify the name as follows.

```
<?evaluator name="mvel"?>
<window id="w" title="MVEL Demo">
    ${new org.zkoss.zul.Textbox().setParent(w)}
</window>
```

Here are a list of built-in implementations.

class

[Optional] [Default: *dependind on how xel-config is specified*]

The implementation used to evaluate the XEL expressions. In addition to the name attribute, you can specify the class directly. For example, you can use MVEL by specifying class as follows.

```
<?evaluator class="org.zkoss.zkmax.xel.mvel.MVELFactory"?>
<window id="w" title="MVEL Demo">
    ${new org.zkoss.zul.Textbox().setParent(w)}
</window>
```

import

[Optiona] [Default: *what are defined in taglib*]

Specifies a list of classes separated with comma to import for evaluating the expression in this page. For example, with MVEL:

```
<?evaluator class="org.zkoss.zkmax.xel.mvel.MVELFactory"
import="org.zkoss.zul.Datebox,org.zkoss.zul.Combobox"?>
<window id="w" title="MVEL Demo">
    ${new Datebox().setParent(w)}
</window>
```

Notice that not all evaluators support the import of classes. For example, all EL-based the evaluators, including the system default one, do *not* support it. In other words, the `import` attribute is meaningless to them (since they don't have the concept of instantiation).

In addition, the class's names specified in the import attribute must be a fully qualified name (including the package's name). In other words, it ignores the classes imported by the import directive.

Version History

Version	Date	Content
6.0.0	September 2011	Support those new features seen in Unified Expression Language 2.2 such as method calls and l-value.

forward

Syntax:

```
<?forward uri="..." [if="..."] [unless="..."]?>
```

It specifies the URI to forward the request to, and the condition to decide whether to forward. If the condition is satisfied or not specified, this page won't be rendered, and the request is, instead, forwarded to the URI specified in the `uri` attribute.

Notes

- Even if the forward is effective (i.e., ZK forwards the request to the specified URI), the initiators specified in the `init` directives will still be called.
- The `createComponents` method of the Execution interface ignores the `forward` directives. In other words, the `forward` directives are evaluated only if the ZUML page is loaded directly.

uri

```
[Required][EL expressions allowed]
```

The URI of the page/servlet to forward to. It may be another ZUML page, a JSP page or any servlet.

If an EL expression is specified and it is evaluated to an empty string, it is considered as no forwarding at all.

if

```
[Optional][Default: true][EL expressions allowed]
```

The condition to forward to the specified URI. If both `if` and `unless` are omitted, this page won't be evaluated and ZK always forwards the request to the specified URI.

unless

```
[Optional][Default: false][EL expressions allowed]
```

The condition *not* to forward to the specified URI. If both `if` and `unless` are omitted, this page won't be evaluated and ZK always forwards the request to the specified URI.

Version History

Version	Date	Content
---------	------	---------

function-mapper

Syntax:

```
<?function-mapper class="..."
  [arg0="..."] [arg1="..."] [arg2="..."] [arg3="..."]?>
```

Specifies the function mapper that will be used by the EL expressions to resolve unknown functions. The specified class must implement the `FunctionMapper`^[1] interface.

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

Notice that the `function-mapper` directives are evaluated before the `init` directives.

class

[Optional]

A class name must implement the `FunctionMapper`^[1] interface. Unlike the `init` directive, the class name cannot be the class that is defined in zscript codes.

arg0, arg1...

[Optional]

You could specify any number of arguments. If not specified, the default constructor is assumed. If specified, it will look for the constructor with the signature in the following order:

1. `Foo(Map args)`
2. `Foo(Object[] args)`
3. `Foo()`

If the first signature is found, the arguments with the name and value are passed to the constructor as an instance of `Map`. If the second signature is found, the values of arguments are passed to the constructor as an array of objects. For example,

```
<?function-mapper class="foo.Foo" whatever="anything"?>
```

Prior to ZK 3.6.2, only the second signature is checked if one or more argument is specified, and it assumes `arg0` as the first argument, `arg1` as the second, and so on.

Version History

Version	Date	Content
---------	------	---------

References

[1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/FunctionMapper.html#>

header

Syntax:

```
<?header name="..." value="..." [append="true|false"] [if="..."] [unless="..."] ?>  
[since 5.0.2]
```

Specifies a response header. It has the same effect as the invocation of `java.lang.String` `Execution.setResponseHeader(java.lang.String, java.lang.String)` ^[1].

name

Required

Specifies the name of the header, such as Pragma.

value

Required, EL allowed

Specifies the value of the header. The value could be an instance of `string` or `Date` (`java.util.Date`).

append

Optional, EL allowed
Default: `false`

Specifies whether to append a response header or to replace (aka., `set`). By default, it is `false`. It means that the previous header with the same name will be replaced. If you want to append the value to the previous value, specify it to `true`.

Version History

Version	Date	Content
---------	------	---------

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#setResponseHeader\(java.lang.String](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#setResponseHeader(java.lang.String),

import

Syntax of Import Class

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

```
[6.0.0]
```

It imports a class or a package of classes. It works like Java's import statement. For example,

```
<?import class="com.foo.composer.FooComposer"?>
<?import class="com.foo.init.*"?>

<?init class="FooInit"?><!-- it will look for com.foo.init.FooInit -->
<window apply="FooComposer"><!-- com.foo.composer.FooComposer will be used -->
...

```

Syntax of Import Directives

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

It imports the directives, such as component definitions (`<?component?>`) and initiators (`<?init?>`), defined in another ZUML 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" macroURI="/WEB-INF/macros/special.zuml" class="Special"?>
<?component name="another" macroURI="/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 directives 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 directives are imported, they won't be changed until the importing page is change, no matter the imported page is changed or not.

class

[Required if importing a class]

The name of a class, or a wildcard, such as `com.foo.app.*`.

uri

[Required if importing directives]

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

directives

[Optional]

If the `directives` attribute is omitted, only the `component`, `init` and `import` (with `class`) directives are imported. If you want to import particular directives, you can specify a list of the names of the directives separated by comma. For example,

```
<?import uri="/template/taglibs.zul" directives="taglib, xel-method"?>
<?import uri="/template/java.zul" directives="import"?><!-- only <?import class="..."?>
```

The directives that can be imported include `component`, `init`, `meta`, `taglib`, `variable-resolver`, and `xel-method`. If you want to import them all, specify `*` to the `directives` attribute. Notice that `meta` implies both the `meta`, `link` and `script` directives.

Version History

Version	Date	Content
6.0.0	July, 2011	The import of classes was introduced.

init

Syntax:

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

It defines an initiator that will be instantiated and called when the ZUML document is loaded.

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 `Execution` ^[1].

You could specify any number of the `init` directive. The specified class must implement the `Initiator` ^[2] interface.

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

Since 3.6.2, you can use any (readable) name instead of `arg0` and so on. For example,

```
<?init class="org.zkoss.zkplus.databind.AnnotateDataBinderInit" root="./abc"?>
```

Then, `java.util.Map` `Initiator.doInit(org.zkoss.zk.ui.Page, java.util.Map)` ^[3] will be called with a map, which contains an entry, whose name is `root` and value `./abc`.

If you'd like to apply an initiator for every page, you don't need to specify it on every page. Rather, you could install a system-level initiator. For more information, please refer to *ZK Developer's Reference: System-level Initiators*.

class

[Optional]

A class name must implement the `Initiator` ^[2] 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. Since 3.6.2, you can use any name for the arguments, but, in the prior version, the first argument must be named as

`arg0`, the second is `arg1` and so on.

Version History

Version	Date	Content
---------	------	---------

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Initiator.html#>
- [3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Initiator.html#doInit\(org.zkoss.zk.ui.Page,](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Initiator.html#doInit(org.zkoss.zk.ui.Page,)

link

Syntax:

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

[since 3.6.2]

It specifies an element that should be generated inside the HEAD element. It is generated *after* ZK default JavaScript and CSS files. Thus, it could override ZK default CSS. Currently only HTML-based clients (so-called browsers) support them. Furthermore, HTML LINK tag is actually generated for each of this declaration.

Developers can specify whatever attributes you like; it is up to the browser to interpret. ZK only encodes the URI of the `href` and `src` attribute (by the use of the `encodeURIComponent` 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.

```
<syntax lang="xml" > <?link rel="alternate" type="application/rss+xml" title="RSS feed"
```

```
  href="/rssfeed.php"?>
```

```
<?link rel="shortcut icon" type="image/x-icon" href="/favicon.ico"?> <?link rel="stylesheet" type="text/css"
href="~/zul/css/ext.css.dsp"?>
```

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

```
</syntax>
```

Alternatives

In addition, you could use the style component to embed JavaScript code. The script component supports more features such as defer, but it has some memory foot print at the server (since it is a component).

Version History

Version	Date	Content
---------	------	---------

meta

Syntax:

```
<?meta [name0="value0"] [name1="value1"] [name2="value2"]
      [if="..."] [unless="..."]?>
```

```
[since 3.6.2]
```

It specifies an element that should be generated inside the HEAD element. It is generated *before* ZK default JavaScript and CSS files. Currently only HTML-based clients (so-called browsers) support them. Furthermore, HTML META tag is actually generated for each of this declaration.

Developers can specify whatever attributes you like; it is up to the browser to interpret. ZK only evaluates the if and unless attributes, and encodes the URI of the href and src 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.

```
<syntax lang="xml" >
```

```
<?meta name="keywords" content="HTML, CSS, XML" ?>
```

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

```
</syntax>
```

Version History

Version	Date	Content
---------	------	---------

page

Syntax:

```
<?page [id="..."] [title="..."] [style="..."] [cacheable="false|true"]
  [language="xul/html"] [zscriptLanguage="Java"]
  [contentType="text/html;charset=UTF-8"]
  [docType="tag PUBLIC "doctype name" "doctype UI"]
  [widgetClass="..."]
  [xml="version="1.0" encoding="UTF-8"]
  [complete="true|false"]
  [automaticTimeout="true|false"]?>
```

It specifies how a page should be handled. The `id` and `title` arguments are the two most important ones.

automaticTimeout

```
[Optional]
[Since 3.6.3]
[Default: the applicatioin default]
```

It specifies whether to automatically redirect to the timeout URI. If it is false, a page will be redirected to the timeout URI when the user takes some action after timeout. In other words, nothing would happen if the user does nothing.

If omitted, whether to automatically timeout depends on the application's configuration. Please refer to ZK Configuration Reference: session-config.

A typical use is to turn off the automatic timeout for the timeout page (otherwise, it will be reloaded again after the session is timeout again). For example,

```
<!-- my timeout page -->
<?page automaticTimeout="false"?>
<zk>
  You didn't access for a while. Please login again.
</zk>
```

Of course, you don't need to do anything, if the timeout page is not a ZUML page.

cacheable

```
[Optional]
[Default: false if Ajax devices, true if XML and MII devices]
```

It specifies whether the client can cache the output.

Note: Browsers, such as Firefox and IE, don't handle the cache of DHTML correctly, so it is not safe to specify `cacheable` with `true` for Ajax devices.

complete

[Optional] [Default: false]

It specifies that this page is a complete page. By complete we mean the page has everything that the client expects. For example, if the client is a HTML browser, then a complete page will generate all the necessary HTML tags, such as `<html>`, `<head>` and `<body>`.

By default (false), a ZK page is assumed to be complete *if and only if* it is *not* included by other page. In other words, if a ZK page is included by other page, ZK will generate `<div>` (if the client is a HTML browser) to enclose the output of the (incomplete) ZK page.

If you have a ZK page that contains a complete HTML page and is included by other pages, you have to specify `true` for this option. For example, the includer is nothing but it includes another page:

```
//includer.jsp
<jsp:include page="includee.zhtml"/>
```

And, the included page contains a complete HTML page:

```
<?page complete="true"?>
<html xmlns="http://www.zkoss.org/2005/zk/native">
  <head>
    <title>My Title</title>
  </head>
  <body>
    My Content
  </body>
</html>
```

contentType

[Optional]
[Default: *depends on the device*]

It specifies the content type. If not specified, it depends on the device. For Ajax devices, it is `text/html; charset=UTF-8`. For XML and MIL devices, it is `text/xml; charset=UTF-8`.

Application developers rarely need to change it, unless for XML devices.

The encoding charset specified here only affects the output of a ZUML document. For browsers, it is the HTML page which receives. The encoding of the JavaScript files or CSS files that a HTML page might reference are not affected.

docType

[Optional]

[Default: *depends on the device*]

It specifies the DOCTYPE (the root tag and DTD) that will be generated to the output directly. This directive is mainly used by XML devices. You rarely need to specify the DOCTYPE directive for Ajax or MIL devices. For example,

```
<?page docType="svg PUBLIC &quot;-//W3C//DTD SVG 1.1//EN&quot; &quot;http://www.w3.org/G
```

will cause the output to be generated with the following snippet

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/s
```

Notice that the `<!DOCTYPE...>` specified in a ZUML page is processed by ZK Loader. It is not part of the output.

If an empty string is specified, DOCTYPE won't be generated (since 5.0.5):

```
<?page docType="" ?>
```

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}"?>
```

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.

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%"?>
```

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 `Page` ^[1] interface.

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

widgetClass

[Options]
[Default: *depending on the device*]
[EL allowed]
[Since 5.0.5]

Specifies the widget class of this page. If not specified, the device's default is assumed. For example, the Ajax device's default is `Page` ^[2].

```
<?page widgetClass="foo.MyPage"?>
```

xml

[Optional]
[Default: *none*]

Specifies the `xml` processing instruction (i.e., `<?xml?>`) that will be generated to the output. Currently only XML devices support this option.

For example,

```
<?page xml="version=&quot;1.0&quot;; encoding=&quot;UTF-8&quot;"?>
```

will generate the following as the first line of the output

```
<?xml version="1.0" encoding="UTF-8"?>
```

zscriptLanguage

[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 zscriptLanguage="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. Please refer to ZUML Reference for more details.

Version History

Version	Date	Content
5.0.5	October, 2010	If empty, DOCTYPE won't be generated.
5.0.5	October, 2010	The widgetClass attribute was introduced.

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Page.html#>
- [2] <http://www.zkoss.org/javadoc/latest/jsdoc/zk/Page.html#>

root-attributes

Syntax:

```
<?root-attributes any-name1="any-value2" any-name2="any-value2"?>
```

It specifies the additional attributes for the root element of the generated output, which depends on the device types. Currently, only Ajax devices support this feature and the root element is the `html` tag. In other words, the attributes specified in the `root-attribute` directives will become the attributes of the `html` element of the generated output. For example,

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

will cause the HTML output to be generated with the following snippet

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

Note: `xmlns="http://www.w3.org/1999/xhtml"`^[1] 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.

any-name="any-value"

Any numbers of names and values are allowed. The value could contain EL expressions.

Version History

Version	Date	Content
---------	------	---------

References

- [1] <http://www.w3.org/1999/xhtml>

script

Syntax:

```
<?script [type="text/javascript"] [src="uri"] [charset="encoding"]
  [content="javascript snippet"] [if="..."] [unless="..."]?>
```

```
[since 3.6.2]
```

It specifies an element that shall be generated inside the HEAD element. It is generated *after* ZK default JavaScript and CSS files. Thus, it could override what is defined in ZK default JavaScript code. Currently only HTML-based clients (so-called browsers) support it. Furthermore, HTML SCRIPT tag is actually generated for each of this declaration.

Developers can specify whatever attributes you like; it is up to the browser to interpret. ZK only evaluates the `if` and `unless` attributes, and encodes the URI of the `href` and `src` attribute (by use of `Execution.encodeURL(java.lang.String)`^[1]). 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.

```
<syntax lang="xml" > <?script src="/js/foo.js"?> <?script content="var foo = true; if (zk.ie) doSomething();"?>
```

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

```
</syntax>
```

As shown above, the attribute value could span multiple lines.

Alternatives

Alternatively, you could use the `script` component to embed JavaScript code. The `script` component supports more features such as `defer`, but it has some memory foot print at the server (since it is a component).

Version History

Version	Date	Content
---------	------	---------

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#encodeURL\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#encodeURL(java.lang.String))

style

Syntax:

```
<?style [type="text/css"] [src="uri"] [charset="encoding"]
  [content="css snippet"] [if="..."] [unless="..."]?>
```

```
[since 5.0.8]
```

It specifies an element that shall be generated inside the HEAD element. It is generated *after* ZK default JavaScript and CSS files. Thus, it could override what is defined in ZK default CSS code. Currently only HTML-based clients (so-called browsers) support it. Furthermore, HTML STYLE or LINK tags are actually generated for each of this declaration.

Developers can specify whatever attributes you like; it is up to the browser to interpret. ZK only evaluates the `if` and `unless` attributes, and encodes the URI of the `href` and `src` attribute (by use of `Execution.encodeURL(java.lang.String)`^[1]). 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.

```
<syntax lang="xml" > <?style src="/css/foo.css"?> <?style content="
```

```
div.blue {background: blue}
```

```
"?>
```

```
My content
```

```
</syntax>
```

As shown above, the attribute value could span multiple lines.

Alternatives

Alternatively, you could use the style component to embed CSS code. Using the style component if you'd like to add or remove the style dynamically (since it is a component), or the page will be included by others.

Version History

Version	Date	Content
5.0.8	July, 2011	The style directive was introduced.

taglib

Syntax:

```
<?taglib uri="myURI" prefix="my"?>
```

This directive is used to load a `taglib` file, which defines a set of static methods that can be used in EL expressions (so called EL functions).

For example, we could load the functions defined in the built-in TLD files identified as `http://www.zkoss.org/dsp/web/core`, and then use the `l` function.

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

If you want to load a TLD file from your Web application, you can specify the path directly. For example, suppose you have a custom TLD at `/WEB-INF/tld/my.tld`, then you could specify it as follows.

```
<?taglib uri="/WEB-INF/tld/my.tld" prefix="my"?>
<listbox>
  <listitem label="{each.name}" forEach="{my:getCustomers()}" />
  <!-- assume there is a function called getCustomers -->
</listbox>
```

The syntax of a `taglib` document is described in the subsection:

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

Version History

Version	Date	Content
---------	------	---------

Custom Taglib

The Syntax of Taglib Document

The syntax of a Taglib document is the same as JSP's taglib (aka., TLD), so you could use JSP's TLD files directly. However, ZK only recognizes the function elements. All others are ignored.

Here is an example:

```
<taglib>
  <function>
    <name>browser</name>
    <function-class>org.zkoss.web.fn.ServletFns</function-class>
    <function-signature>
boolean isBrowser(java.lang.String)
    </function-signature>
    <description>
Whether the current request is coming from the browser of the
specified
type.
    </description>
  </function>
  <function>
    <name>l</name>
    <function-class>org.zkoss.xel.fn.CommonFns</function-class>
    <function-signature>java.lang.String getLabel(java.lang.String)</function-signature>
    <description>
Returns the label of the specified key.
    </description>
  </function>
</taglib>
```

where

- The root element must be called taglib
- Each function declaration must be called function. It requires three sub-elements: name, function-class and function-signature. The description element is optional (for documentation only).

In addition, you could import all public static methods with an element called import, and the name of EL function will be the same as the method name. For example,

```
<import>
  <import-name>Labels</import-name>
  <import-class>org.zkoss.util.resource.Labels</import-class>
</import>
```

Configure Tag Documents as Built-in

The custom taglib document could be specified as follows (assuming you have a taglib called /WEB-INF/tld/foo.tld):

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

In addition, you could map a taglib to URL as if they are built-in. First, provide a file named /META-INF/tld/config.xml that can be found in the classpath. For example, you could put it under WEB-INF/classes/META-INF/tld/config.xml, or as part of a JAR file. Then, in this file (config.xml), you could specify any number of the mapping as follows.

```
<config>
  <taglib>
    <taglib-uri>http://www.foo.com/myfirst</taglib-uri>
    <taglib-location>/whatever/myfirst.tld</taglib-location>
  </taglib>
  <taglib>
    <taglib-uri>http://www.foo.com/mysecond</taglib-uri>
    <taglib-location>/whatever/mysecond.tld</taglib-location>
  </taglib>
</config>
```

Notice that the location must be a path accessible by the classpath (such as /WEB-INF/classes or a JAR file).

Then, you could use them as follows.

```
<?taglib uri="http://www.foo.com/myfirst" prefix="f"?>
<?taglib uri="http://www.foo.com/mysecond" prefix="s"?>
```

Version History

Version	Date	Content
---------	------	---------

variable-resolver

Syntax:

```
<?variable-resolver class="..."
  [arg0="..."] [arg1="..."] [arg2="..."] [arg3="..."]?>
```

Specifies the variable resolver that could be used by the `zscript` interpreter and the EL expressions to resolve unknown variables. The specified class must implement the `VariableResolver`^[4] 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"?>
```

Notice that if you have a variable resolver used for every page, you don't have to declare it on every page. Rather, you could register a system-level variable resolver. For more information, please refer to ZK Developer's Reference: System-level Variable Resolvers.

class

[Required]

A class name that must implement the `VariableResolver`^[4] interface. Unlike the `init` directive, the class name cannot be the class that is defined in `zscript` codes.

arg0, arg1...

[Optional]

You could specify any number of arguments. If not specified, the default constructor is assumed. If specified, it will look for the constructor with the signature in the following order:

1. `Foo(Map args)`
2. `Foo(Object[] args)`
3. `Foo()`

If the first signature is found, the arguments with the name and value are passed to the constructor as an instance of `Map`. If the second signature is found, the values of arguments are passed to the constructor as an array of objects.

```
<?variable-resolver class="foo.Foo" whatever="anything"?>
```

will cause `Foo(Map args)` being called with a map, which has an entry: `whatever=anything`. If not found, `Foo(Object[] args)` will be called with a single-item array and the value of the item is `anything`.

Prior to ZK 3.6.2, only the second signature is checked if one or more argument is specified, and it assumes `arg0` as the first argument, `arg1` as the second, and so on.

Version History

Version	Date	Content
---------	------	---------

xel-method

Syntax:

```
<?xel-method prefix="..." name="..." class="..." signature="..."?>
```

Specifies an EL function that could be used in EL expressions. For example,

```
<?xel-method prefix="c" name="forName"
  class="java.lang.Class"
  signature="java.lang.Class forName(java.lang.String)"?>
<textbox value="\${c:forName('java.util.List')}" />
```

prefix

[Required]

Specifies the prefix used to identify this method.

name

[Required]

Specifies the name used to identify this method. The full name is "prefix:name".

class

[Required]

Specifies the class that the method is defined in.

signature

[Required]

The signature of the method. Note: the method must be public static. In additions, Java 5 Generics are *not* allowed.

Version History

Version	Date	Content
---------	------	---------

EL Expressions

The syntax of an EL expressions is `${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 as a Boolean object.

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

Associate with Java

There are several ways to associate Java objects with EL expressions.

1. Implement a variable resolver (`VariableResolver` ^[4]) and specify it with the `variable-resolver` directive.
2. Return the object in a static method and specify it in the `xel-method`
3. Declare multiple static methods in a `taglib` and declare it in `taglib`
4. Construct them in `zscript`

Here is the detailed information for each feature. For introductory, please refer to [ZK Developer's Reference](#).

Literals

EL expressions define the following literals:

Type	Description
Boolean	true and false
Integer	as in Java, such as 123
Floating point	as in Java, such as 1.23 and 1e9
String	with single and double quotes; " is escaped as \", ' is escaped as \', and \ is escaped as \\. Example, 'a string' and "hello world"
Null	null

Version History

Version	Date	Content
---------	------	---------

Operators

EL expressions provide the following operators^[1]:

Type	Operators	Description
Arithmetic	+, - ^[2] , *, /, div, %, mod, - ^[3]	<ul style="list-style-type: none"> / and div are the same, while % and mod are the same.
Logical	and, &&, or, , not, !, empty	<ul style="list-style-type: none"> The empty operator is a prefix operation that can be used to determine whether a value is null or empty, such as <code>#{empty foo}</code>.
Relational	==, eq, !=, ne, <, lt, >, gt, <=, ge, >=, le	<ul style="list-style-type: none"> Comparisons can be made against other values, or against boolean, string, integer, or floating point literals.
Conditional	A ? B : C	It evaluate B or C, depending on the result of the evaluation of A.
Index	[]	<p>To evaluate <code>expr-a[expr-b]</code>, evaluate <code>expr-a</code> into <code>value-a</code> and evaluate <code>expr-b</code> into <code>value-b</code>. If either <code>value-a</code> or <code>value-b</code> is null, return null.</p> <ul style="list-style-type: none"> If <code>value-a</code> is a Map, return <code>value-a.get(value-b)</code>. If <code>!value-a.containsKey(value-b)</code>, then return null. If <code>value-a</code> is a List or array, coerce <code>value-b</code> to int and return <code>value-a.get(value-b)</code> or <code>Array.get(value-a, value-b)</code>, as appropriate. If the coercion couldn't be performed, an error is returned. If the get call returns an <code>IndexOutOfBoundsException</code>, null is returned. If the get call returns another exception, an error is returned. If <code>value-a</code> is a JavaBeans object, coerce <code>value-b</code> to String. If <code>value-b</code> is a readable property of <code>value-a</code>, then return the result of a get call. If the get method throws an exception, an error is returned.
Member	.	<ul style="list-style-type: none"> Properties of variables are accessed using the <code>.</code> operator and can be nested arbitrarily. The value of a map can be accessed by using the <code>.</code> operator.

[1] The information is from JSP Tutorial (<http://download.oracle.com/javase/1.4/tutorial/doc/JSPIntro7.html>).

[2] binary

[3] unary

The relative precedence levels of operators from the highest to lowest, left to right are as follows:

- [] .
- ()^[1]
- -^[2] not ! empty
- * / div % mod
- + -^[3]
- < > <= >= lt gt le ge
- == != eq ne
- && and
- || or
- ? :

[1] Used to change the precedence of operators.

[2] unary

[3] binary

Version History

Version	Date	Content
---------	------	---------

Type Coercion

EL expressions will coerce the type automatically. Here is the summary of the coercion rules.

Note: The coercion is also applied to function arguments. For example, `{c:doSomething(null)}` will cause `doSomething("")` being called if it expects a String object.

	Boolean	Character	Number	String
Boolean	<code>obj</code> ^[1]	ERROR	ERROR	<code>obj.toString()</code>
Character	ERROR	<code>obj</code>	<code>(short)obj</code>	<code>obj.toString()</code>
Number	<code>obj</code>	ERROR	<code>(char)obj</code>	<code>obj.toString()</code>
String (not empty)	<code>Boolean.valueOf(obj)</code>	<code>obj.charAt(0)</code>	<code>Number</code> ^[2] <code>.valueOf(x)</code>	<code>obj</code>
String (empty)	<code>false</code>	<code>(char)0</code>	<code>0</code>	<code>"" (obj)</code>
null	<code>false</code>	<code>(char)0</code>	<code>0</code>	<code>""</code>
Other	ERROR	ERROR	ERROR	<code>obj.toString()</code>

[1] `obj` represents the object being coerced

[2] The real class is determined at run time, such as Integer and Float.

- The handling of an empty string and null is the same

Version History

Version	Date	Content
---------	------	---------

Implicit Objects

EL expressions define a set of implicit objects that you can access directly in an EL expression. Here is a complete list of implicit objects.

applicationScope

applicationScope - java.util.Map

A map of custom attributes associated with the Web application. It is the same as the `getAttributes` method in the `WebApp`^[1] 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 `<mp>javax.servlet.ServletContext</mp>`. In other words, you could use it to communicate with other servlets, such as JSF.

Version History

Version	Date	Content
---------	------	---------

References

[1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/WebApp.html#>

arg

arg - java.util.Map

The `arg` argument passed to the `createComponents` method in the `Executions` ^[1] 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 `CreateEvent` ^[2] class.

Version History

Version	Date	Content
---------	------	---------

References

[1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#>

[2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/CreateEvent.html#>

componentScope

componentScope - java.util.Map

A map of custom attributes associated with the component. It is the same as the `getAttributes` method in the `Component`^[1] interface.

Version History

Version	Date	Content
---------	------	---------

cookie

cookie - java.util.Map

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

Version History

Version	Date	Content
---------	------	---------

desktop

desktop - Desktop ^[1]

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

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

Version History

Version	Date	Content
---------	------	---------

References

[1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Desktop.html#>

desktopScope

desktopScope - java.util.Map

A map of custom attributes associated with the desktop. It is the same as the `getAttributes` method in the `Desktop` ^[1] interface.

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

Version History

Version	Date	Content
---------	------	---------

each

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>
```

Nested forEach

To retrieve the index of the iteration, or the previous `each` object in nested `forEach`, you have to use another implicit object called `forEachStatus`.

```
<listbox forEach="{matrix}">
  <listitem label="{forEachStatus.previous.each.label}: {each}" forEach="{each.items}" />
</listbox>
```

In Java

You could access the `each` object directly in `zscript` such as:

```
<window>
  <button label="{each}" forEach="apple, orange">
    <zscript>
      self.parent.appendChild(new Label("" + each));
    </zscript>
  </button>
</window>
```

The `each` object is actually stored in the parent component's attribute, so you could retrieve it in pure Java as follows.

```
public class Foo implements Composer {
  public void doAfterCompose(Component comp) throws Exception {
    Object each = comp.getParent().getAttribute("each"); //retrieve
    the each object
    ForEachStatus forEachStatus =
    (ForEachStatus) comp.getParent().getAttribute("forEachStatus");
    //...
  }
}
```

If the component is a root, you could retrieve them from page's attributes (`Page.getAttribute(java.lang.String)`^[1]).

However, the value of `each` is reset after the XML element that `forEach` is associated has been evaluated. Thus, you cannot access it in an event listener, unless you store the value first. For more information, please refer to ZK Developer's Reference: Iterative Evaluation.

Version History

Version	Date	Content
---------	------	---------

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Page.html#getAttribute\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Page.html#getAttribute(java.lang.String))

event

event - Event ^[1] or derived

The current event. It is available for the event listener only.

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

Version History

Version	Date	Content
---------	------	---------

References

[1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Event.html#>

execution

execution – Execution ^[1]

The current execution.

Version History

Version	Date	Content
---------	------	---------

header

header - java.util.Map

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

Version History

Version	Date	Content
---------	------	---------

headerValues

headerValues - java.util.Map

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

Version History

Version	Date	Content
---------	------	---------

forEachStatus

forEachStatus – ForEachStatus ^[1]

The status of an iteration. It is an instance of ForEachStatus ^[1]. ZK exposes the information relative to the iteration taking place when evaluating the iterative element.

```
<listbox width="100px">
  <listitem label="{forEachStatus.index}: {each}" forEach="Best, Better, Good"/>
</listbox>
```

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.

To retrieve the information of the outer iterator if an iteration is nested, you could use `ForEachStatus.getPrevious()` ^[1].

```
<listbox forEach="{matrix}">
  <listitem label="{forEachStatus.previous.each.label}: {each}" forEach="{each.items}"
</listbox>
```

Version History

Version	Date	Content
---------	------	---------

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ForEachStatus.html#getPrevious\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ForEachStatus.html#getPrevious())

labels

labels - java.util.Map

A map of all internationalization labels belonging to the current locale (`Locales.getCurrent()`^[1]).

For example, if you have a property file as follows:

```
owner=Foo Inc.
application.name=Killer
application.title=Killer 2011
```

Then, you could access them with this implicit object as follows.

```
<grid>
  <row>${labels.owner}</row>
  <row>${labels.application.name}</row>
  <row>${labels.application.title}</row>
</grid>
```

Notice that the key of a property could be name as *key1.key2*, and EL expressions could retrieve them correctly. More precisely, ZK groups the segmented labels as map. For example, `${labels.app}` was resolved as a map containing two entries (title and description).

```
app.title=Foo
app.description=A super application
```

If you have a key named as the prefix of the other keys, you have to use `$` to access it. For example, `${labels.app.$}` is required to resolve the label with key named `app`.

```
app=Application
app.title=Foo
app.description=A super application
```

Under the hood: The `labels` object is actually the map returned by `Labels.getSegmentedLabels()`^[2]. Furthermore, if the key of a property contains dot (`.`), all properties with the same prefix are grouped as another map. For example, `${labels.application}` (i.e., `Labels.getSegmentedLables().get("application")`) will return a map containing two entries (name and title) in the previous example.

Version History

Version	Date	Content
5.0.7	March, 2011	This implicit object was introduced.

References

- [1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/Locales.html#getCurrent\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/Locales.html#getCurrent())
 [2] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/resource/Labels.html#getSegmentedLabels\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/resource/Labels.html#getSegmentedLabels())

page

page - Page ^[1]

The current page.

Version History

Version	Date	Content
---------	------	---------

pageContext

pageContext – PageContext ^[1]

The current page context used to retrieve the request, response, variable resolver and so on.

Version History

Version	Date	Content
---------	------	---------

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/web/servlet/xel/PageContext.html#>
-

pageScope

pageScope - java.util.Map

A map of custom attributes associated with the current page. It is the same as the `getAttributes` method in the `Page`^[1] interface.

Version History

Version	Date	Content
---------	------	---------

param

param - java.util.Map

A map of parameters of the request, `Map<String, String>`.

To retrieve all possible parameter values, use `paramValues` instead.

```
`${param.something}`  
`${paramValues.something[0]}`
```

Notice that, in `zscript`, there is no `paramValues`. `Param` is a map of possible values, `Map<String, String[]>`.

```
<zscript>  
String[] values = param.get("something");  
</zscript>
```

Version History

Version	Date	Content
---------	------	---------

paramValues

paramValues - java.util.Map

A map of parameters of the request, `Map<String, String[]>`.

To retrieve the first value of a parameter if any, use `param` instead.

```
${param.something}
${paramValues.something[1]}
```

Notice that, in `zscript`, there is no `paramValues`. `Param` is a map of possible values, `Map<String, String[]>`.

```
<zscript>
String[] values = param.get("something");
</zscript>
```

Version History

Version	Date	Content
---------	------	---------

requestScope

requestScope – java.util.Map

A map of custom attributes associated with the current execution. It is the same as `getAttributes` method in the `Execution` ^[1] interface.

Version History

Version	Date	Content
---------	------	---------

self

self - Component ^[1]

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>
```

Version History

Version	Date	Content
---------	------	---------

session

session - Session ^[1]

The session. It is similar to `javax.servlet.http.HttpSession`^[2].

Version History

Version	Date	Content
---------	------	---------

sessionScope

sessionScope - java.util.Map

A map of custom attributes associated with the session. It is the same as the `getAttributes` method in the `Session` ^[1] 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.

Version History

Version	Date	Content
---------	------	---------

spaceOwner

spaceOwner - IdSpace ^[1]

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

Version History

Version	Date	Content
---------	------	---------

References

[1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/IdSpace.html#>

spaceScope

spaceScope - java.util.Map

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

Version History

Version	Date	Content
---------	------	---------

Core Methods

In this section we describe the EL functions defined in the built-in TLD called <http://www.zkoss.org/dsp/web/core>.

For example,

```
<?taglib uri="http://www.zkoss.org/dsp/web/core" prefix="c"?>

<window title="${c:l('app.title')}">
  ...
</window>
```

EL functions in this TLD file are described in the following subsections.

boolean

```
boolean boolean(Object value);
```

i.e., `CommonFns.toBoolean(java.lang.Object)` ^[1]

Converts the specified object to a boolean.

Notice that EL will convert the type automatically, so you rarely need this method. Please refer to Type Coercion for details.

Parameters:

- value - the value to convert

Version History

Version	Date	Content
---------	------	---------

References

- [1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/CommonFns.html#toBoolean\(java.lang.Object\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/CommonFns.html#toBoolean(java.lang.Object))

browser

```
boolean browser(String type);
```

i.e., `ServletFns.isBrowser(java.lang.String)` ^[1]

Returns if the current request comes from the browser of the specified type.

Parameters:

- type - the type of the browser.

Allowed values include "robot", "ie", "ie6", "ie6-", "ie7", "ie8", "ie9", "ie7-", "ie8-", "gecko", "gecko2", "gecko3", "gecko2-", "opera", "safari"

Note: "ie6-" means Internet Explorer 6 only; not Internet Explorer 7 or other.

Version History

Version	Date	Content
---------	------	---------

References

- [1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/web/fn/ServletFns.html#isBrowser\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/web/fn/ServletFns.html#isBrowser(java.lang.String))

cat

```
String cat(String s1, String s2);
```

i.e., `java.lang.String`) `StringFns.cat(java.lang.String, java.lang.String)` ^[1]

Concatenates two strings. Note: null is considered as empty.

Version History

Version	Date	Content
---------	------	---------

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/StringFns.html#cat\(java.lang.String,](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/StringFns.html#cat(java.lang.String,)

cat3

```
String cat3(String s1, String s2, String s3);
```

i.e., `java.lang.String, java.lang.String`) `StringFns.cat3(java.lang.String, java.lang.String, java.lang.String)` ^[1]

Concatenates three strings together. Note: null is considered as empty.

Version History

Version	Date	Content
---------	------	---------

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/StringFns.html#cat3\(java.lang.String,](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/StringFns.html#cat3(java.lang.String,)

cat4

```
String cat4(String s1, String s2, String s3, String s4);
```

i.e., java.lang.String, java.lang.String, java.lang.String) StringFns.cat4(java.lang.String, java.lang.String, java.lang.String, java.lang.String) ^[1]

Concatenates four strings together. Note: null is considered as empty.

Version History

Version	Date	Content
---------	------	---------

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/StringFns.html#cat4\(java.lang.String](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/StringFns.html#cat4(java.lang.String),

cat5

```
String cat5(String s1, String s2, String s3, String s4, String s5);
```

i.e., java.lang.String, java.lang.String, java.lang.String, java.lang.String) StringFns.cat5(java.lang.String, java.lang.String, java.lang.String, java.lang.String, java.lang.String) ^[1]

Concatenates five strings together. Note: null is considered as empty.

Version History

Version	Date	Content
---------	------	---------

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/StringFns.html#cat5\(java.lang.String](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/StringFns.html#cat5(java.lang.String),

char

```
char char(Object value);
```

i.e., `CommonFns.toChar(java.lang.Object)` ^[1]

Converts the specified object to a character.

Notice that EL will convert the type automatically, so you rarely need this method. Please refer to Type Coercion for details.

Parameters:

- value - the value to convert

Version History

Version	Date	Content
---------	------	---------

References

- [1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/CommonFns.html#toChar\(java.lang.Object\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/CommonFns.html#toChar(java.lang.Object))

class

```
Class class(String className);
```

i.e., `Classes.forNameByThread(java.lang.String)` ^[1]

Returns the class of the given class name.

Version History

Version	Date	Content
---------	------	---------

References

- [1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/lang/Classes.html#forNameByThread\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/lang/Classes.html#forNameByThread(java.lang.String))

decimal

```
BigDecimal decimal(Object value);
```

i.e., `CommonFns.toDecimal(java.lang.Object)` ^[1]

Converts the specified object to a big decimal.

Parameters:

- value - the value to convert

Version History

Version	Date	Content
---------	------	---------

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/CommonFns.html#toDecimal\(java.lang.Object\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/CommonFns.html#toDecimal(java.lang.Object))

encodeURL

```
String encodeURL(String uri);
```

i.e., `ServletFns.encodeURL(java.lang.String)` ^[1]

Encodes a URL.

If an URI contains "*", it will be replaced with a proper Locale. For example, if the current Locale is zh_TW and the resource is named "ab*.cd", then it searches "ab_zh_TW.cd", "ab_zh.cd" and then "ab.cd", until any of them is found.

Note: "*" must be right before ".", or the last character. For example, "ab*.cd" and "ab*" are both correct, while "ab*cd" and "ab*Vcd" are ignored.

If an URI contains two "*", the first "*" will be replaced with a browser code and the second with a proper locale. The browser code depends on what browser the user are used to visit the web site. Currently, the code for Internet Explorer is "ie", Safari is "saf", Opera is "opr" and all others are "moz". Thus, in the above example, if the resource is named "ab**.cd" and Firefox is used, then it searches "abmoz_zh_TW.cd", "abmoz_zh.cd" and then "abmoz.cd", until any of them is found.

Parameters:

- uri - the URI to encode
-

Version History

Version	Date	Content
---------	------	---------

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/web/fn/ServletFns.html#encodeURL\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/web/fn/ServletFns.html#encodeURL(java.lang.String))

endsWith

```
boolean endsWith(String value, String suffix);
```

i.e., `java.lang.String`) `StringFns.endsWith(java.lang.String, java.lang.String)` ^[1]

Tests if this string ends with the specified suffix.

Parameters:

- value - the value to test
- suffix - the suffix to test

Version History

Version	Date	Content
5.0.7	March, 2011	This method was introduced

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/StringFns.html#endsWith\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/StringFns.html#endsWith(java.lang.String)).

escapeXML

```
String escapeXML(String s);
```

i.e., XMLs.escapeXML(java.lang.String)^[1]

Encodes a string that special characters are quoted to be compatible with HTML/XML. For example,

< is translated to <

> to >

& to &

" to "

' to '

Parameters:

- s - the string to encode

Version History

Version	Date	Content
---------	------	---------

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xml/XMLs.html#escapeXML\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xml/XMLs.html#escapeXML(java.lang.String))

getCurrentLocale

```
java.util.Locale getCurrentLocale();
```

i.e., Locales.getCurrent()^[1]

Returns the locale for the current request (or thread); never null.

Version History

Version	Date	Content
---------	------	---------

indexOf

```
int indexOf(Object value, Object element);
```

i.e., `java.lang.Object`) `CommonFns.indexOf(java.lang.Object, java.lang.Object)` ^[1]

Returns the index within the value of the first occurrence of the specified element.

Parameters:

- `value` - the value to test. If it is an instance of `String`, `String.indexOf()` is called. If it is a collection or an array, all of its elements are examined one-by-one. If it is a map, `Map.keySet()` will be examined one-by-one.
- `element` - the element to test.

Version History

Version	Date	Content
5.0.7	March, 2011	This method was introduced

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/CommonFns.html#indexOf\(java.lang.Object\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/CommonFns.html#indexOf(java.lang.Object)),

int

```
int int(Object value);
```

i.e., `CommonFns.toInt(java.lang.Object)` ^[1]

Converts the specified object to an integer.

Notice that EL will convert the type automatically, so you rarely need this method. Please refer to Type Coercion for details.

Parameters:

- `value` - the value to convert

Version History

Version	Date	Content
---------	------	---------

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/CommonFns.html#toInt\(java.lang.Object\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/CommonFns.html#toInt(java.lang.Object))

isInstance

```
Class isInstance(Object class, Object value);
```

i.e., `java.lang.Object`) `CommonFns.isInstance(java.lang.Object, java.lang.Object)` ^[1]

Tests whether an object (the second argument, value) is an instance of a class (the first argument, class). You could specify a class or the class name as the first argument.

Version History

Version	Date	Content
---------	------	---------

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/CommonFns.html#isInstance\(java.lang.Object,](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/CommonFns.html#isInstance(java.lang.Object,)

join

```
String join(Object[] value, String separator);
```

i.e., ^[1], `java.lang.String`) `StringFns.join(java.lang.Object[], java.lang.String)`

Joins the given array of values into a string and separated with the given separator

Parameters:

- value - the array of values to join (if any of its elements is not a string, it will be converted to a string first)
- separator - the separator

Version History

Version	Date	Content
5.0.7	March, 2011	This method was introduced

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/StringFns.html#join\(java.lang.Object](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/StringFns.html#join(java.lang.Object)

l

```
String l(String key);
```

i.e., `CommonFns.getLabel(java.lang.String)` ^[1]

Returns the label of the given key defined in the internationalization labels.

The label is based on the current Locale (`Locales.getCurrent()` ^[1]).

For 5.0.7 and later, an implicit object called `labels` was introduced, and it is more convenient to use.

For example,

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

Version History

Version	Date	Content
---------	------	---------

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/CommonFns.html#getLabel\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/CommonFns.html#getLabel(java.lang.String))

l2

```
String l2(String key, Object[] args);
```

i.e., `java.lang.Object[1]` `CommonFns.getLabel(java.lang.String, java.lang.Object[])`

Returns the label of the given key defined in the internationalization labels, and formats it with the given arguments. The formatting is done by the use of `MessageFormat[2]`.

The label is based on the current `Locale` (`Locales.getCurrent()[1]`).

For example, let us assume we want to generate a full name based on the current `Locale`, then we could use `${c:l2('key', args)}` to generate concatenated messages as follows.

```
<?taglib uri="http://www.zkoss.org/dsp/web/core" prefix="c"?>
<label value="${c:l2('fullname.format', fullname)}">
```

where we assume `fullname` is a string array (such as `new String[] {"Jimmy", "Shiau"}).`

Version History

Version	Date	Content
---------	------	---------

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/CommonFns.html#getLabel\(java.lang.String](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/CommonFns.html#getLabel(java.lang.String),

[2] <http://download.oracle.com/javase/6/docs/api/java/text/MessageFormat.html>

lastIndexOf

```
int lastIndexOf(Object value, Object element);
```

i.e., `java.lang.Object`) `CommonFns.lastIndexOf(java.lang.Object, java.lang.Object)` ^[1]

Returns the index within the value of the last occurrence of the specified element.

Parameters:

- `value` - the value to test. If it is an instance of `String`, `String.lastIndexOf()` is called. If it is a list or an array, all of its elements are examined one-by-one. It does not support `Collection` and `Map`.
- `element` - the element to test.

Version History

Version	Date	Content
5.0.7	March, 2011	This method was introduced

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/CommonFns.html#lastIndexOf\(java.lang.Object\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/CommonFns.html#lastIndexOf(java.lang.Object)),

length

```
int length(Object value);
```

i.e., `CommonFns.length(java.lang.Object)` ^[1]

Returns the length of a string, array, collection or map.

Notice that there is a built-in operator to test if a string/array/collection/map is empty:

```
${empty foo}
```

Version History

Version	Date	Content
---------	------	---------

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/CommonFns.html#length\(java.lang.Object\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/CommonFns.html#length(java.lang.Object))

number

```
Number number(Object value);
```

i.e., `CommonFns.toNumber(java.lang.Object)` ^[1]

Converts the specified object to a number.

Notice that EL will convert the type automatically, so you rarely need this method. Please refer to Type Coercion for details.

Parameters:

- `value` - the value to convert

Version History

Version	Date	Content
---------	------	---------

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/CommonFns.html#toNumber\(java.lang.Object\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/CommonFns.html#toNumber(java.lang.Object))

property

```
String property(String key);
```

i.e., `Library.getProperty(java.lang.String)` ^[1]

Returns the value of the given library property, or null if not found.

Version History

Version	Date	Content
---------	------	---------

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/lang/Library.html#getProperty\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/lang/Library.html#getProperty(java.lang.String))

new

```
Object new(Object cls);
```

i.e., `CommonFns.new_(java.lang.Object)` ^[1]

Instantiates the given class. It assumes the given class has a default constructor.

Paramters

- `cls` - the class. It could be an instance of either `String` or `Class`.

Version History

Version	Date	Content
5.0.6	December 2010	Automatically converted a number to the correct type (aka., class). Notice that a number specified in EL is interpreted as long, by default. For example, in <code>\${c:new('foo.Mine', 10)}</code> , 10 is interpreted as long. If you're using 5.0.5 and prior, you have to convert it to integer manually: <code>\${c:new('foo.Mine', c:int(10))}</code> .

References

- [1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/CommonFns.html#new_\(java.lang.Object\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/CommonFns.html#new_(java.lang.Object))

new1

```
Object new1(Object cls, Object arg);
```

i.e., `java.lang.Object` `CommonFns.new_(java.lang.Object, java.lang.Object)` ^[1]

Instantiates the given class with an argument. It assumes the given class has a proper constructor.

Paramters

- `cls` - the class. It could be an instance of either `String` or `Class`.
- `arg` - the argument to be passed to the constructor.

Version History

Version	Date	Content
---------	------	---------

References

- [1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/CommonFns.html#new_\(java.lang.Object,](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/CommonFns.html#new_(java.lang.Object,)

new2

```
Object new2(Object cls, Object arg1, Object arg2);
```

i.e., java.lang.Object, java.lang.Object) CommonFns.new_(java.lang.Object, java.lang.Object, java.lang.Object) ^[1]

Instantiates the given class with two arguments. It assumes the given class has a proper constructor.

Parameters

- cls - the class. It could be an instance of either String or Class.
- arg1 - the first argument to be passed to the constructor.
- arg2 - the second argument to be passed to the constructor.

Version History

Version	Date	Content
---------	------	---------

new3

```
Object new3(Object cls, Object arg1, Object arg2, Object arg3);
```

i.e., java.lang.Object, java.lang.Object) CommonFns.new_(java.lang.Object, java.lang.Object, java.lang.Object) ^[1]

Instantiates the given class with three arguments. It assumes the given class has a proper constructor.

Parameters

- cls - the class. It could be an instance of either String or Class.
- arg1 - the first argument to be passed to the constructor.
- arg2 - the second argument to be passed to the constructor.
- arg3 - the third argument to be passed to the constructor.

Version History

Version	Date	Content
---------	------	---------

split

```
String[] split(String value, String regex);
```

i.e., java.lang.String) StringFns.split(java.lang.String, java.lang.String) ^[1]

Splits this string to match the given regular expression.

Parameters:

- value - the value to split
- regex - the delimiting regular expression, such as ";" and "[,;]".

Version History

Version	Date	Content
5.0.7	March, 2011	This method was introduced

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/StringFns.html#split\(java.lang.String](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/StringFns.html#split(java.lang.String),

startsWith

```
boolean startsWith(String value, String prefix);
```

i.e., java.lang.String) StringFns.startsWith(java.lang.String, java.lang.String) ^[1]

Tests if this string starts with the specified prefix.

Parameters:

- value - the value to test
- prefix - the prefix to test

Version History

Version	Date	Content
5.0.7	March, 2011	This method was introduced

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/StringFns.html#startsWith\(java.lang.String](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/StringFns.html#startsWith(java.lang.String),

string

```
String toString(Object value);
```

i.e., `CommonFns.toString(java.lang.Object)` ^[1]

Converts the specified object to a string.

Parameters:

- value - the value to convert

Version History

Version	Date	Content
---------	------	---------

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/CommonFns.html#toString\(java.lang.Object\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/CommonFns.html#toString(java.lang.Object))

substring

```
String substring(String s, int from, int to);
```

i.e., `int, int) StringFns.substring(java.lang.String, int, int)` ^[1]

Returns a new string that is a substring of the given string.

Parameters:

- s - the string to retrieve the substring
- from - the beginning index, inclusive
- to - the ending index, exclusive

Version History

Version	Date	Content
5.0.7	March, 2011	This method was introduced

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/StringFns.html#substring\(java.lang.String,](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/StringFns.html#substring(java.lang.String,)

testCurrentLocale

```
boolean testCurrent(String lang, String country);
```

i.e., `java.lang.String`) `Locales.testCurrent(java.lang.String, java.lang.String)` ^[1]

Returns whether the current locale (`getCurrentLocale()`) belongs to the specified language and/or country.

Parameters:

- lang - the language code, e.g., en and zh. Ignored if null.
- country - the country code, e.g., US. Ignored if null. If empty, it means no country code at all.

Version History

Version	Date	Content
---------	------	---------

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/Locales.html#testCurrent\(java.lang.String,](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/Locales.html#testCurrent(java.lang.String,)

toLowerCase

```
String toLowerCase(String value);
```

i.e., `StringFns.toLowerCase(java.lang.String)` ^[1]

Converts all of the characters in this string to lowercase using the rules of the current Locale (`Locale.getCurrent()` ^[1]).

Parameters:

- value - the value to convert

Version History

Version	Date	Content
5.0.7	March, 2011	This method was introduced

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/StringFns.html#toLowerCase\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/StringFns.html#toLowerCase(java.lang.String))

toUpperCase

```
String toUpperCase(String value);
```

i.e., `StringFns.toUpperCase(java.lang.String)` ^[1]

Converts all of the characters in this string to uppercase using the rules of the current Locale (`Locales.getCurrent()` ^[1]).

Parameters:

- value - the value to convert

Version History

Version	Date	Content
5.0.7	March, 2011	This method was introduced

References

- [1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/StringFns.html#toUpperCase\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/StringFns.html#toUpperCase(java.lang.String))

trim

```
String trim(String value);
```

i.e., `StringFns.trim(java.lang.String)` ^[1]

Returns a copy of the string, with leading and trailing whitespace omitted.

Parameters:

- value - the value to trim

Version History

Version	Date	Content
5.0.7	March, 2011	This method was introduced

References

- [1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/StringFns.html#trim\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/fn/StringFns.html#trim(java.lang.String))
-

Extensions

There are several ways to extend ZUML:

1. Add additional languages (aka., component sets)
2. Add more implicit objects by the use of variable resolvers
3. Add more functions by the use of function mappers or xel-method

This chapter describes additional interpreters, such as Groovy, and additional EL evaluators, such as MVEL.

zscript

The default interpreter for the zscript elements is Java (based on BeanShell ^[1]). Depending on your preference, you could choose one of built-in interpreters, or implement your own interpreter.

The built-in interpreters includes: Java, Groovy, Ruby, Python, and JavaScript.

Choose Interpreter for Whole Page

To change the default interpreter for the whole page, you could use the page directive by specifying the zscriptLanguage attribute, such as

```
<?page zscriptLanguage="Groovy"?>
<window border="normal">
  <vbox id="vb">
    <label id="l" value="Hi"/>
    <button label="change label" onClick="l.value='Hi, Groovy';"/>
    <button label="add label" onClick="new Label('New').setParent(vb);"/>
  </vbox>
  <button label="alert" onClick="alert('Hi, Groovy')"/>
</window>
```

Choose Interpreter for zscript

You could choose an interpreter for a particular zscript element by specifying the language attribute as follows.

```
<zscript language="Ruby">
(Java::Label.new 'New').parent = $vb
</zscript>
```

Choose Interpreter for Event Handler

You could choose an interpreter for a particular event handler by prefixing it with the language name as follows.

```
<button label="alert" onClick="python:alert('Hi, Python')"/>
```

Support More Scripting Languages

Currently ZK supports Java, JavaScript, Ruby, Groovy, and Python. However, it is easy to extend:

1. Provide a class that implements `Interpreter` [2]. However, it is suggested to derive from `GenericInterpreter` [3] for simplicity.
2. Declare the scripting language in either `WEB-INF/zk.xml`, or `zk/config.xml`.

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

Multi-Scope versus Single-Scope

Depending on the implementation, an interpreter might have exactly one logical scope, or one logic scope per ID space to store these variables and methods declared in `zscript`. For the sake of description, we will call them the single-scope and multi-scope interpreters respectively.

For example, ZK's Java interpreter(`BeanShell`) is a multi-Scope Interpreter. On the other hand, Ruby, Groovy and JavaScript interpreters don't support multi-scope. It means all variables defined in, say, Ruby are stored in one logical scope (per interpreter). To avoid confusion, you could prefix the variable names with special prefix denoting the window.

Notice that each page has its own interpreter to evaluate the `zscript` code.

Version History

Version	Date	Content
---------	------	---------

References

- [1] <http://www.beanshell.org>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/scripting/Interpreter.html#>
- [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/scripting/util/GenericInterpreter.html#>

EL Expressions

The default evaluator for EL expressions are derived from Apache Commons EL ^[1]. Thus, its functionality is the same as JSP 2.0's EL expressions^[2].

If you prefer a more powerful EL evaluator, such as MVEL, OGNL^[3] or your own implementation, you could specify it with the evaluator directive. For example,

```
<?evaluator name="mvel"
    import="org.zkoss.zul.Datebox,org.zkoss.zul.Combobox"?>

<window id="w" title="MVEL Demo">
    You see a textbox appended with MVEL:
    ${new Datebox().setParent(w)}
    Another example:
    ${new org.zkoss.zul.Textbox().setParent(w)}
    Another:
    ${new Combobox().setParent(w)}
</window>
```

[1] <http://commons.apache.org/el/>

[2] Notice that the package names are all changed, and the dependency of JSP EL is removed, so it is OK to run under any Web server without any conflict

[3] Both MVEL and OGNL are supported in ZK EE.

Version History

Version	Date	Content
---------	------	---------

Article Sources and Contributors

ZUML Reference *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference *Contributors:* Alicelin, Sphota, Tmillsclare, Tomyeh

ZUML *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML *Contributors:* Tmillsclare, Tomyeh

Languages *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Languages *Contributors:* Alicelin, Tomyeh

ZUL *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Languages/ZUL *Contributors:* Char, Tomyeh

XHTML *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Languages/XHTML *Contributors:* Alicelin, Char, Tomyeh

XML *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Languages/XML *Contributors:* Alicelin, Char, Tomyeh

Namespaces *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Namespaces *Contributors:* Char, Jimmyschau, Tomyeh, Tonyq

Client *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Namespaces/Client *Contributors:* Char, Tomyeh

Client Attribute *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Namespaces/Client_Attribute *Contributors:* Char, Tomyeh

Native *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Namespaces/Native *Contributors:* Tmillsclare, Tomyeh

ZK *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Namespaces/ZK *Contributors:* Char, Tomyeh

Elements *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Elements *Contributors:* Maya001122, Tmillsclare, Tomyeh

attribute *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Elements/attribute *Contributors:* Maya001122, Tmillsclare, Tomyeh

custom-attributes *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Elements/custom-attributes *Contributors:* Maya001122, Tmillsclare, Tomyeh

variables *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Elements/variables *Contributors:* Maya001122, Tmillsclare, Tomyeh

zk *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Elements/zk *Contributors:* Alicelin, Maya001122, Tmillsclare, Tomyeh

zscript *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Elements/zscript *Contributors:* Alicelin, Maya001122, Tmillsclare, Tomyeh

Attributes *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Attributes *Contributors:* Alicelin, Maya001122, Tmillsclare, Tomyeh

apply *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Attributes/apply *Contributors:* Alicelin, Maya001122, Tmillsclare, Tomyeh

forEach *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Attributes/forEach *Contributors:* Alicelin, Maya001122, Tmillsclare, Tomyeh

forEachBegin *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Attributes/forEachBegin *Contributors:* Alicelin, Maya001122, Tmillsclare, Tomyeh

forEachEnd *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Attributes/forEachEnd *Contributors:* Alicelin, Maya001122, Tmillsclare, Tomyeh

forward *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Attributes/forward *Contributors:* Alicelin, Ashishd, Maya001122, Tmillsclare, Tomyeh

fulfill *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Attributes/fulfill *Contributors:* Maya001122, Tmillsclare, Tomyeh

if *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Attributes/if *Contributors:* Maya001122, Tmillsclare, Tomyeh

unless *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Attributes/unless *Contributors:* Maya001122, Tmillsclare, Tomyeh

use *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Attributes/use *Contributors:* Alicelin, Maya001122, Tmillsclare, Tomyeh

Texts *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Texts *Contributors:* Char, Tomyeh

Processing Instructions *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Processing_Instructions *Contributors:* Alicelin, Tomyeh

component *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Processing_Instructions/component *Contributors:* Alicelin, Maya001122, Tmillsclare, Tomyeh

evaluator *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Processing_Instructions/evaluator *Contributors:* Alicelin, Henrichen, Maya001122, Tmillsclare, Tomyeh

forward *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Processing_Instructions/forward *Contributors:* Alicelin, Maya001122, Tmillsclare, Tomyeh

function-mapper *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Processing_Instructions/function-mapper *Contributors:* Alicelin, Maya001122, Tmillsclare, Tomyeh

header *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Processing_Instructions/header *Contributors:* Alicelin, Maya001122, Tmillsclare, Tomyeh

import *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Processing_Instructions/import *Contributors:* Maya001122, Tmillsclare, Tomyeh

init *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Processing_Instructions/init *Contributors:* Alicelin, Flyworld, Maya001122, Tmillsclare, Tomyeh

link *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Processing_Instructions/link *Contributors:* Alicelin, Flyworld, Maya001122, Tmillsclare, Tomyeh

meta *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Processing_Instructions/meta *Contributors:* Alicelin, Tomyeh

page *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Processing_Instructions/page *Contributors:* Alicelin, Char, Maya001122, SimonPai, Tmillsclare, Tomyeh

root-attributes *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Processing_Instructions/root-attributes *Contributors:* Maya001122, Tmillsclare, Tomyeh

script *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Processing_Instructions/script *Contributors:* Tomyeh

style *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Processing_Instructions/style *Contributors:* Tomyeh

taglib *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Processing_Instructions/taglib *Contributors:* Alicelin, Maya001122, Tmillsclare, Tomyeh

Custom Taglib *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Processing_Instructions/taglib/Custom_Taglib *Contributors:* Alicelin, Sphota, Tomyeh

variable-resolver *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Processing_Instructions/variable-resolver *Contributors:* Alicelin, Maya001122, Tmillsclare, Tomyeh

xel-method *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/ZUML/Processing_Instructions/xel-method *Contributors:* Maya001122, Tmillsclare, Tomyeh

EL Expressions *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions *Contributors:* Alicelin, Tomyeh

Literals *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Literals *Contributors:* Tomyeh

Operators *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Operators *Contributors:* Alicelin, Tomyeh

Type Coercion *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Type_Coercion *Contributors:* Tomyeh

Implicit Objects *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Implicit_Objects *Contributors:* Tmillsclare, Tomyeh

applicationScope *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Implicit_Objects/applicationScope *Contributors:* Alicelin, Maya001122, Tmillsclare, Tomyeh

arg *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Implicit_Objects/arg *Contributors:* Maya001122, Tmillsclare, Tomyeh

componentScope *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Implicit_Objects/componentScope *Contributors:* Maya001122, Tmillsclare, Tomyeh

cookie *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Implicit_Objects/cookie *Contributors:* Maya001122, Tmillsclare, Tomyeh

desktop *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Implicit_Objects/desktop *Contributors:* Maya001122, Tmillsclare, Tomyeh

desktopScope *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Implicit_Objects/desktopScope *Contributors:* Maya001122, Tmillsclare, Tomyeh

each *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Implicit_Objects/each *Contributors:* Maya001122, Tmillsclare, Tomyeh

event *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Implicit_Objects/event *Contributors:* Alicelin, Maya001122, Tmillsclare, Tomyeh

execution *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Implicit_Objects/execution *Contributors:* Maya001122, Tmillsclare, Tomyeh

header *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Implicit_Objects/header *Contributors:* Maya001122, Tmillsclare, Tomyeh

headerValues *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Implicit_Objects/headerValues *Contributors:* Maya001122, Tmillsclare, Tomyeh

forEachStatus *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Implicit_Objects/forEachStatus *Contributors:* Alicelin, Maya001122, Tmillsclare, Tomyeh

labels *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Implicit_Objects/labels *Contributors:* Alicelin, Tomyeh

page *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Implicit_Objects/page *Contributors:* Maya001122, Tmillsclare, Tomyeh

pageContext *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Implicit_Objects/pageContext *Contributors:* Maya001122, Tmillsclare, Tomyeh

pageScope *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Implicit_Objects/pageScope *Contributors:* Maya001122, Tmillsclare, Tomyeh

param *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Implicit_Objects/param *Contributors:* Alicelin, Maya001122, Tmillsclare, Tomyeh

paramValues *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Implicit_Objects/paramValues *Contributors:* Alicelin, Maya001122, Tmillsclare, Tomyeh

requestScope *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Implicit_Objects/requestScope *Contributors:* Maya001122, Tmillsclare, Tomyeh

self *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Implicit_Objects/self *Contributors:* Maya001122, Tmillsclare, Tomyeh

session *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Implicit_Objects/session *Contributors:* Maya001122, Tmillsclare, Tomyeh

sessionScope *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Implicit_Objects/sessionScope *Contributors:* Maya001122, Tmillsclare, Tomyeh

spaceOwner *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Implicit_Objects/spaceOwner *Contributors:* Maya001122, Tmillsclare, Tomyeh

spaceScope *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Implicit_Objects/spaceScope *Contributors:* Maya001122, Tmillsclare, Tomyeh

Core Methods *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods *Contributors:* Tomyeh

boolean *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/boolean *Contributors:* Char, Tomyeh

browser *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/browser *Contributors:* Alicelin, Char, SimonPai, Tomyeh

cat *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/cat *Contributors:* Char, Tomyeh

cat3 *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/cat3 *Contributors:* Char, Tomyeh

cat4 *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/cat4 *Contributors:* Char, Tomyeh

cat5 *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/cat5 *Contributors:* Char, Tomyeh

char *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/char *Contributors:* Char, Tomyeh

class *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/class *Contributors:* Char, Tomyeh

decimal *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/decimal *Contributors:* Char, Tomyeh

encodeURL *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/encodeURL *Contributors:* Char, Tomyeh

endsWith *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/endsWith *Contributors:* Tomyeh

escapeXML *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/escapeXML *Contributors:* Char, Tomyeh

getCurrentLocale *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/getCurrentLocale *Contributors:* Char, Tomyeh

indexOf *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/indexOf *Contributors:* Tomyeh

int *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/int *Contributors:* Char, Tomyeh

isInstance *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/isInstance *Contributors:* Char, Tomyeh

join *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/join *Contributors:* Tomyeh

l *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/l *Contributors:* Char, Tomyeh

l2 *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/l2 *Contributors:* Alicelin, Tomyeh

lastIndexOf *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/lastIndexOf *Contributors:* Tomyeh

length *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/length *Contributors:* Char, Tomyeh

number *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/number *Contributors:* Char, Tomyeh

property *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/property *Contributors:* Char, Tomyeh

new *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/new *Contributors:* Char, Tomyeh

new1 *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/new1 *Contributors:* Char, Tomyeh

new2 *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/new2 *Contributors:* Char, Tomyeh

new3 *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/new3 *Contributors:* Char, Tomyeh

split *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/split *Contributors:* Alicelin, Tomyeh

startsWith *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/startsWith *Contributors:* Tomyeh

string *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/string *Contributors:* Char, Tomyeh

substring *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/substring *Contributors:* Tomyeh

testCurrentLocale *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/testCurrentLocale *Contributors:* Char, Tomyeh

toLowerCase *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/toLowerCase *Contributors:* Alicelin, Tomyeh

toUpperCase *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/toUpperCase *Contributors:* Alicelin, Tomyeh

trim *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/EL_Expressions/Core_Methods/trim *Contributors:* Tomyeh

Extensions *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/Extensions *Contributors:* Alicelin, Tomyeh

zscript *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/Extensions/zscript *Contributors:* Alicelin, Char, Tomyeh

EL Expressions *Source:* http://books.zkoss.org/index.php?title=ZUML_Reference/Extensions/EL_Expressions *Contributors:* Alicelin, Char, Tomyeh
