SIMPLY RICH

# ZK™

# The Developer's Guide

**Version 1.2.0**

May 2006

**Potix Corporation**

Revision 61

# Table of Contents

# 1. Introduction

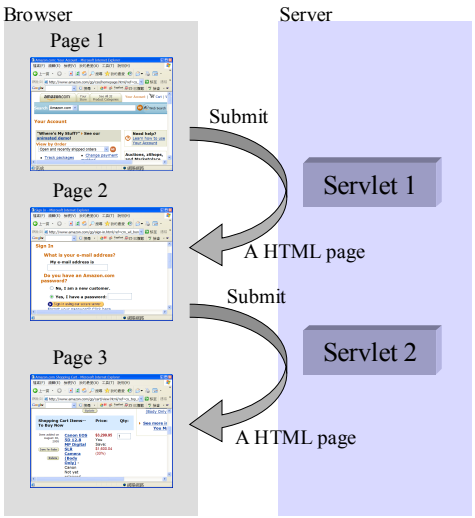
Welcome to ZK, the simplest way to make Web applications rich.

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

The chapter describes the historical background about Web programming, AJAX technologies and the ZK project. You might skip this chapter if you prefer to get familiar with the ZK features directly.

## Traditional Web Applications

Aiming at exchanging documents simply and effectively, Web technologies, HTTP and HTML, is originated from the page-based and stateless-communication model. In this model, a page is self-contained and the minimal unit to communicate between clients and servers.



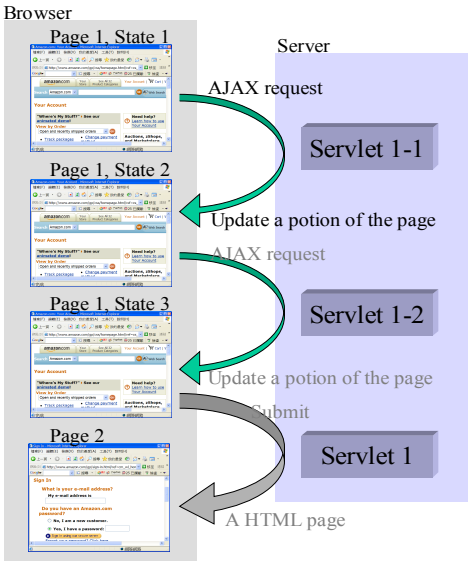

As the Web has emerged as the default platform for application development, this model faces a substantial challenge: the inability to visually represent the complexities in today's applications. For example, to give a customer a quotation, you might have to open another page to search his trading records, another page for the recent prices, and another page for current stocking. Users are forced to leave the page he is working on, and navigate among several pages. It is easy to get lost and confused, and the result is unhappy customers, lost sales and low productivities.

The challenge to develop a modern application upon this page-based model is also substantial. In this model, applications running at the server have to take care everything from parsing the request, rendering the response, routing processes that link users from one page to another, and handling versatile errors made by users. Tens of frameworks, such as Struct, Tapestry and JSF, are then emerged to simplify this development process. Due to the huge gap between the page-based model and the modern applications, learning and using these frameworks is never a pleasant process, not to mention intuition or simplicity.

## Ad-hoc AJAX Applications

Over a decade of evolution, Web applications evolved from static HTML pages, to Dynamic HTML

pages, to applets and Flash, and, finally, to AJAX[1] technologies (Asynchronous JavaScript and XML). Illustrated by Google Maps and Suggest, AJAX breaths new life into Web applications by delivering the same level of interactivity and responsiveness as desktop applications. Unlike applets or Flash, AJAX is based the standard browser and JavaScript and no proprietary plugin is required.




AJAX is a kind of new generation DHTML. Like DHTML, it heavily relies on JavaScript to listen events triggered by user's activity, and then manipulate visual representation of a page (aka. DOM) in the browser dynamically. Moreover, it takes a step further by enabling the communication with the server asynchronously without leaving or rendering the whole page again. It breaks the page-based model by introducing light-weight communication between clients and servers. With proper design, AJAX could bring rich components common to desktop applications to life in Web applications, and all of their content could be dynamically updated under the control of applications.

When providing the interactivity that users demand, AJAX adds more complexities and skill prerequisites to the already costly development of Web applications. Developers have to manipulate DOM in the browser and communicate with the server in incompatible and even buggy JavaScript API. For better interactivity, developers have to replicate subset of application data and business logic to the browser. It then increases the maintenance cost and the challenge to synchronized data in between.

The bottom line is that ad hoc AJAX applications is no different from traditional Web applications regarding the way to process requests. Developers still have to fulfill the gap caused by the page-based and stateless model.

## ZK: What It Is

ZK is an event-driven, component-based framework to enable rich user interfaces for Web applications. ZK includes an AJAX-based event-driven engine, a rich set of XUL and XHTML components, and a markup language called ZUML (ZK User Interface Markup Language).

With ZK, you represent your application in feature-rich XUL and XHTML components, and manipulate them upon events triggered by user's activity, as you did for years in desktop applications. Unlike most of other frameworks, AJAX is a behind-the-scene technology. The synchronization of the content of components and the pipelining of events are done automatically by the ZK engine.

Your users get the same engaged interactivity and responsiveness as a desktop application, while

1 AJAX is coined by Jesse James Garrett in Ajax: A New Approach to Web Applications.

your development remains the same simplicity as that of desktop applications.

In addition to a simple model and rich components, ZK also supports a markup languages, called ZUML. ZUML, like XHTML, enables developers to design user interfaces without programming. With XML namespaces, ZUML seamlessly integrates different set of tags[2] into the same page. Currently, ZUML supports two set of tags, XUL and HTML.

For fast prototyping and customization, ZUML allows developers to embed Java[3] and EL expressions. Developers could choose not to embed Java codes at all if they prefer a more rigid discipline. Unlike JavaScript embedded in HTML, ZK executes all embedded codes in the server.

It is interesting to note what we said everything running at the server is from the viewpoint of application developers. For component developers, they have to balance the interactivity and simplicity by deciding what tasks being done at the browser, what at the server.

## ZK: What It Is Not

ZK assumed nothing about persistence or inter-server communication. ZK is designed to be as thin as possible. It is only aimed at the presentation tier. It does not require or suggest any other back-end technologies. All your favorite middlewares work as they used to, such as JDBC, Hibernate, Java Mail, EJB or JMS.

ZK doesn't provide a tunnel, RMI or other API for developers to communicate between clients and servers, because all codes are running at the server at the same JVM.

ZK doesn't enforce developers to use MVC or other design patterns. Whether to use them is the developer's choice.

ZK is not a framework aiming to bring XUL to Web applications. It is aimed to bring the desktop programming model to Web applications. Currently, it supports XUL and XHTML. In future, it might support XAML, XQuery and others.

ZK embedded AJAX in the current implementation. It doesn't end in where AJAX ends. With upcoming ZK for Mobile, your applications could reach any devices that support J2ME, such as PDA, mobiles and game consoles. Moreover, you don't need to modify your application at all[4].

## ZK: Limitations

ZK is not for applications that run most of tasks at the clients, such as 3D action games.

Unless you write a special component, ZK is not for applications that want to leverage the computing power at the clients.

2 A tag is an XML element. When a ZUML page is interpreted, a corresponding component is created.

3 The interpretation of Java is done by BeanShell (http://www.beanshell.org).

4 For devices with small screen, you usually have to adjust the presentation pages.

# 2. Getting Started

This chapter describes how to write your first ZUML page. It is suggested to read at least this chapter, if you are in hurry.

This chapter uses XUL to illustrate ZK features, but it is usually applicable to other markup languages that ZK supports.

## Hello World!

After ZK is installed into your favorite Web server[5], writing applications is straight forward. Just create a file, say hello.zul, as follows[6] under a proper directory.

```
<window title="Hello" border="normal">
   Hello World!
</window>
```

Then, browse to the right URL, say http://localhost/myapp/hello.zul, and you got it.


In a ZUML page, a XML element describes what component to create. In this example, it is a window (`com.potix.zul.html.Window`). The XML attributes are used to assign values to properties of the window component. In this example, it creates a window with a title and border, which is done by setting the `title` and `border` properties to "Hello" and "normal", respectively.

The text enclosed in the XML elements is also interpreted as a special component called `label` (`com.poitx.zul.html.Label`). Thus, the above example is equivalent to the following.

```
<window title="Hello" border="normal">
   <label value="Hello World!"/>
</window>
```

## Interactivity

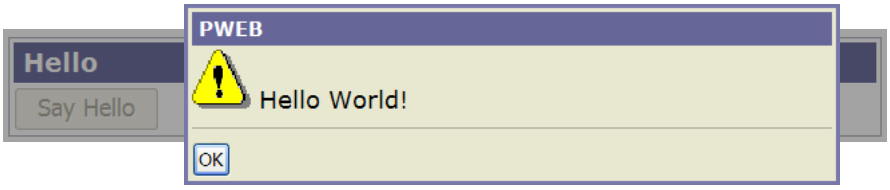
Let us put some interactivity into it.

```
<window title="Hello" border="normal">
   <button label="Say Hello" onClick="alert(&quot;Hello World!&quot;)"/>
</window>
```

Then, when you click the button, you see as follows.

5 Refer to the Quick Start Guide.

6 The other way to try examples depicted here is to use the live demo to run them.




The `onClick` attribute is a special attribute used to add an event listener to the component. The attribute value could be any legal Java codes. Notice that we use `&quot;` to denote the double quot (") to make it a legal XML document. If you are not familiar with XML, you might take a look at the **XML** section in the **ZK User Interface Markup Language** chapter.

The `alert` function is a global function to display a message dialog box. It is a shortcut to one of the `show` methods of the `com.potix.zul.html.Messagebox` class.

```
<button label="Say Hello" onClick="Messagebox.show(&quot;Hello World!&quot;)"/>
```

**Notes:**

- The scripts embedded in ZUML pages are all Java and they are running at the server. They are *not* JavaScript.
- ZK uses BeanShell to interpret Java at run time, so you could declare global functions, such as `alert`, for it.
- All classes in the `java.lang`, `java.util`, `com.potix.zk.ui`, `com.potix.zk.ui.event` and `com.potix.zul.html` package are imported before evaluating the Java codes embedded in ZUML pages.

## The `zscript` Element

The `zscript` element is a special element to defines Java codes to be evaluated when a ZUML page is rendered. It is usually used to do initialization, or to declare global functions.

> **Note:** You cannot use EL expressions in `zscript` codes. On the other hand, variables declared in EL expressions are visible to the same ID space (so accessible by EL expressions). Refer to the **ID Space** section in the **Basics** chapter.

For example, the following example displays a different message each time the button is pressed.

```
<window title="Hello" border="normal">
   <button label="Say Hello" onClick="sayHello()"/>
   <zscript>
   int count = 0;
   void sayHello() { //declare a global function
      alert("Hello World! "+ ++count);
   }
   </zscript>
</window>
```

**Note:** `zscript` is evaluated only once when the page is loaded. It is usually used to define routines and initial variables.

### The `src` Attribute

To separate codes and views, developers could put Java codes in a separated file, say `sayHello.zs`, and then use the `src` attribute to reference it.

```
<window title="Hello" border="normal">
   <button label="Say Hello" onClick="sayHello()"/>
   <zscript src="sayHello.zs"/>
</window>
```

which assumes the content of `sayHello.zs` is as follows.

```
int count = 0;
void sayHello() { //declare a global function
   alert("Hello World! "+ ++count);
}
```

## The `attribute` Element

The `attribute` element is a special element to define a XML attribute of the enclosing element. With proper use, it makes the page more readable. The following is equivalent to `hello.zul` described above.

```
<button label="Say Hello">
    <attribute name="onClick">alert("Hello World!");</attribute>
</button>
```

## The EL Expressions

Like JSP, you could use EL expressions in any part of ZUML pages, except the names of attributes, elements and processing instructions.

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

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

**Tip:** `empty` is an operator used to test whether a map, a collection, an array or a string is null or empty.

**Tip:** `map[entry]` is a way to access an element of a map. In other words, it is the same as `map.get(entry)` in Java.

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

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

> **Tip:** The + operator in EL is arithmetic. It doesn't handle string catenations. If you want to catenate strings, simple use "`${expr1} is added with ${expr2}`".

Standard implicit objects, such as `param` and `requestScope`, and ZK implicit objects, such as `self` and `page`, are supported to simplify the use.

```
<textbox value="${param.who} does ${param.what}"/>
```

To import EL functions from TLD files, you could use a processing instruction called `taglib` as follows.

```
<?taglib uri="/WEB-INF/tld/web/core.tld" prefix="p" ?>
```

The **Developer's Reference** provides more details on EL expressions. Or, you might refer to JSP 2.0 tutorials or guides for more information about EL expressions.

## The `id` Attribute

To access a component in Java codes and EL expressions, you could assign an identifier to it by use of the `id` attribute. In the following example, we set an identifier to a label such that we could manipulate its value when one of the buttons is pressed.

```
<window title="Vote" border="normal">
   Do you like ZK? <label id="label"/>
   <separator/>
   <button label="Yes" onClick="label.value = self.label"/>
   <button label="No" onClick="label.value = self.label"/>
</window>
```

After pressing the Yes button, you will see the following.


The following is any example for referencing a component in an EL expression.

```
<textbox id="source" value="ABC"/>
<label value="${source.value}"/>
```

## The `if` and `unless` Attributes

The `if` and `unless` attributes are used to control whether to create a component. In the following examples, both labels are created only if the request has a parameter called `vote`.

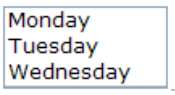
```
<label value="Vote 1" if="${param.vote}"/>
<label value="Vote 2" unless="${!param.vote}"/>
```

If both attributes are specified, the component won't be created unless they are both evaluated to true.

## The `forEach` Attribute

The `forEach` attribute is used to control how many components shall be created. If you specify a collection of objects to this attribute, ZK Loader will create a component for each item of the specified collection. For example, in the following ZUML page, the listitem element will evaluated three times (for "Monday", "Tuesday" and "Wednesday") and then generate three list items.

```
<zscript>contacts = new String[] {"Monday", "Tuesday", "Wednesday"};</zscript>
<listbox width="100px">
  <listitem label="${each}" forEach="${contacts}"/>
</listbox>
```




When evaluating the element with the `forEach` attribute, the `each` variable is assigned one-by-one with objects from the collection, i.e., `contacts` in the previous example. Thus, the above ZUML page is the same as follows.

```
<listbox>
  <listitem label="Monday"/>
  <listitem label="Tuesday"/>
  <listitem label="Wednesday"/>
</listbox>
```

## The `use` Attribute

Embedding codes improperly in pages might cause maintenance headache. There are two ways to separate codes from views.

First, you could listen to events you care, and then invoke the proper methods accordingly. For example, you could invoke your methods to initialize, process and cancel upon the `onCreate`[7], `onOK`[8] and `onCancel`[9] events.

7 The onCreate event is sent when a window defined in a ZUML page is created.
8 The onOK event is sent when user pressed the ENTER key.
9 The onCancel event is sent when user pressed the ESC key.

```
<window id="main" onCreate="MyClass.init(main)"
   onOK="MyClass.process(main)" onCancel="MyClass.cancel(main)"/>
```

In addition, you must have a Java class called MyClass shown as follows.

```
import com.potix.zul.html.Window;

public class MyClass {
   public static void init(Window main) { //does initialization
   }
   public static void save(Window main) { //saves the result
   }
   public static void cancel(Window main) { //cancel any changes
   }
}
```

Second, you could use the `use` attribute to specify a class to replace the default component class.

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

Then, you must have a Java class called MyWindow as follows.

```
import com.potix.zul.html.Window;

public class MyWindow extends Window {
   public void onCreate() { //does initialization
   }
   public void onOK() { //save the result
   }
   public void onCancel() { //cancel any changes
   }
}
```

These two approaches have different advantages. They both act as the controller in the MVC paradigm. The choice is yours.

### Implement Java Classes in `zscript`

Thanks to the power of BeanShell[10], the implementation of Java classes can be done in `zscript` as follows.

```
<zscript>
   public class MyWindow extends Window {
   }
</zscript>
<window use="MyWindow"/>
```

To separate codes from the view, you can put all `zscript` codes in a separated file, say `mywnd.zs`, and then,

```
<zscript src="/zs/mywnd.zs"/>
```

10 http://www.beanshell.org

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

**Tip:** You can use the `init` directive to specify a `zscript` file, too. The difference is the `init` directive is evaluated before any component is created (in the Page Initial phase). For more information, refer to the **`init` Directive** section in the **ZK User Interface Markup Language** chapter.

## Create Components Manually

In addition to describe what components to create in ZUML pages, developers could create them manually. All component classes are concrete. You create them directly[11] with their constructors.

```
<window id="main">
   <button label="Add Item">
      <attribute name="onClick">
   new Label("Added at "+new Date()).setParent(main);
   new Separator().setParent(main);
      </attribute>
   </button>
   <separator bar="true"/>
</window>
```

When a component is created manually, it won't be added to any page automatically. In other words, it doesn't appear at user's browser. To add it to a page, you could invoke the `setParent`, `appendChild` or `insertBefore` method to assign a parent to it, and it becomes a part of a page if the parent is a part of a page.

There is no destroy or close method for components[12]. A component is removed from the browser as soon as it is detached from the page. It is shown as soon as it is attached to the page.

```
<window id="main">
   <zscript>Component detached = null;</zscript>
   <button id="btn" label="Detach">
      <attribute name="onClick">
   if(detached != null) {
      detached.setParent(main);
      detached = null;
      btn.label = "Detach";
   } else {
      (detached = target).setParent(null);
      btn.label = "Attach";
   }
      </attribute>
   </button>
   <separator bar="true"/>
```

11 To make things simpler, the factory design pattern is not used.

12 The concept is similar to W3C DOM. On the other hand, Windows API required developers to manage the lifecycle.

```
   <label id="target" value="You see this if it is attached."/>
</window>
```

In the above example, you could use the `setVisible` method to have a similar effect. However, `setVisible(false)` doesn't remove the component from the browser. It just makes a component (and all its children) invisible.

After a component is detached from a page, the memory it occupies is release by JVM's garbage collector if the application has no reference to it.

## Define New Components for a Particular Page

As illustrated, it is simple to assign properties to a component by use of XML attributes.

```
<button label="OK" style="border:1px solid blue"/>
```

ZK provides a powerful yet simple way to let developers define new components for a particular pages. It is useful if most components of the same type share a set of properties.

First, you use the `component` directive to define a new component.

```
<?component name="bluebutton" extends="button" style="border:1px solid blue" label="OK"?>

<bluebutton/>
<bluebutton label="Cancel"/>
```

is equivalent to

```
<bluebutton style="border:1px solid blue" label="OK"/>
<bluebutton  style="border:1px solid blue" label="Cancel"/>
```

Moreover, you can override the definition of `button` altogether as follows. Of course, it won't affect any other pages.

```
<?component name="button" extends="button" style="border:1px solid blue" label="OK"?>

<button/>
<button label="Cancel"/>
```

For more information, refer to the **`component` Directive** section in the **ZK User Interface Markup Language** chapter.

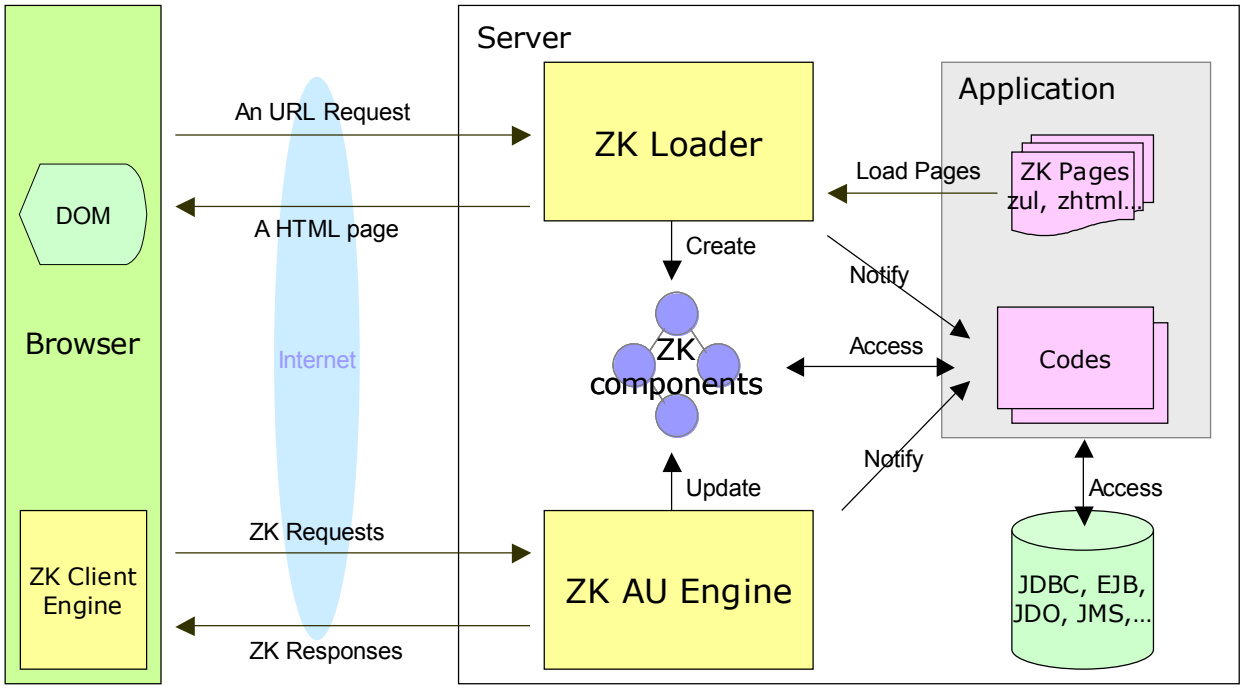
# 3. The Basics

This chapter describes the basics of ZK. It uses XUL to illustrate ZK features, but it is usually applicable to other markup languages that ZK supports.

## Architecture Overview

ZK includes an AJAX-based mechanism to automate interactivity, a rich set of XUL-based components to enrich usability, and a markup language to simplify development.

The AJAX-based mechanism consists of three parts as depicted below: ZK loader, ZK AU Engine and ZK Client Engine.




Based on the user's request, the ZK Loader loads a ZK page, interprets it, and renders the result into HTML pages in response to URL requests. A ZK page is written in a markup language called ZUML. ZUML, like HTML, is used to describe what components to create and how to represent them visually. These components, once created, remain available until the session is timeout.

The ZK AU[13] Engine and the ZK Client Engine then work together as pitcher and catcher. They deliver events happening in the browser to the application running at the server, and update the DOM tree at the browser based on how components are manipulated by the application. This is so-called event-driven programming model.

13 AU stands for Asynchronous Update.

### The Execution Flow

1. When a user types an URL or clicks an hyperlink at the browser, a request is sent to the Web server. ZK loader is then invoked to serve this request, if the URL matches which ZK is configured for[14].
2. ZK loader loads the specified page and interprets it to create proper components accordingly.
3. After interpreting the whole page, ZK loader renders the result into a HTML page. The HTML page is then sent back to the browser accompanied with ZK Client Engine[15].
4. ZK Client engine sits at the browser to detect any event triggered by user's activity such as moving mouse or changing a value. Once detected, it notifies ZK AU Engine by sending a ZK request[16].
5. Upon receiving ZK requests from Client Engine, AU Engine updates the content of corresponding component, if necessary. And then, AU Engine notifies the application by invoking relevant event handlers, if any.
6. If the application chooses to change content of components, add or move components, AU Engine send the new content of altered components to Client Engine by use of ZK responses.
7. These ZK responses are actually commands to instruct Client Engine how to update the DOM tree accordingly.

## Components, Pages and Desktops

### Components

A component is an UI object, such as a label, a button and a tree. It defines the visual representation and behaviors of a particular user interface. By manipulating them, developers control how to represent an application visually in the client.

A component must implement the `com.potix.zk.ui.Component` interface.

### Pages

A page (`com.potix.zk.ui.Page`) is a collection of components. A page confines components belonging to it, such that they will be displayed in a certain portion of the browser. A page is automatically created when ZK loader interprets a ZUML page.

14 Refer to **Appendix A**.

15 ZK Client Engine is written in JavaScript. Browsers cache ZK Client engine, so the engine is usually sent only once at the first visit.

16 ZK requests are special AJAX requests. However, for the mobile edition, ZK requests are special HTTP requests.

### Page Title

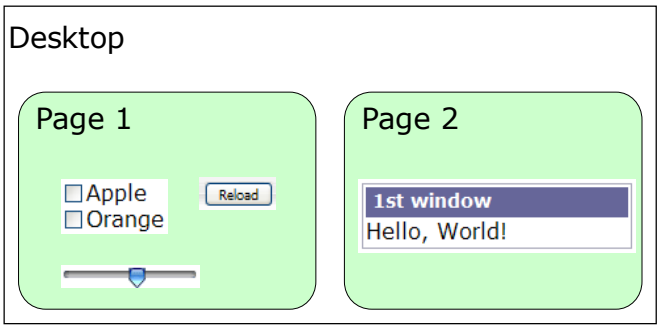
Each page could have a title that will be displayed as part of the browser's window caption. Refer to the **Processing Instructions** section in the **ZK User Interface Markup Language** chapter for details.

```
<?page title="My Page Title"?>
```

## Desktops

A ZUML page might include another ZUML pages directly or indirectly. Since these pages are created for serving the same URL request, they are collectively called a desktop (`com.potix.zk.ui.Desktop`). In other word, a desktop is a collection of pages for serving the same URL request.

As a ZK application interacts with user, more pages might be added to a desktop and some might be removed from a desktop. Similarly, a component might be added to or removed from a page.




### The `createComponents` Method

Notice that both pages and desktops are created and remove implicitly. There are no API to create or remove them. A page is create each time ZUML loads a page. A page is removed when ZK finds it is no longer referenced. A desktop is created when the first ZUML page is loaded. A desktop is removed if too many desktops are created for the specific session.

The `createComponents` method in the `com.potix.zk.ui.Executions` class creates only components, *not page*, even though it loads a ZUML file (aka., page).

## Forest of Trees of Components

A component has at most one parent. A component might have multiple children. Some components accept only certain types of components as children. Some must be a child of certain type of components. Some don't allow any child at all. For example, Listbox in XUL accepts Listcols and Listitem only. Refer to Javadoc or XUL tutorials for details.

A component without any parent is called a root component. A page might have multiple root components, which could be retrieved by the `getRoots` method.

### Component: a Visual Representation and a Java Object

Besides being a Java object in the server, a component has a visual part[17] in the browser, if and only if it belongs to a page. When a component is attached to a page, its visual part is created[18]. When a component is detached from a page, its visual part is removed.

There are two ways to attach a component into a page. First, you could call the `setPage` method to make a component to become a root component of the specified page. Second, you could call the `setParent`, `insertBefore` or `appendChild` method to make it to become a child of another component. Then, the child component belongs to the same page as to which the parent belongs.

Similarly, you could detach a root component from a page by calling `setPage` with `null`. A child is detached if it is detached from a parent or its parent is detached from a page.

### Identifiers

Each component has an identifier (the `getId` method). It is created automatically when a component is created. Developers could change it anytime. There is no limitation about how an identifier shall be named. However, if an alphabetical identifier is assigned, developers could access it directly in Java codes and EL expression embedded in the ZUML page.

```
<window title="Vote" border="normal">
   Do you like ZK? <label id="label"/>
   <separator/>
   <button label="Yes" onClick="label.value = self.label"/>
   <button label="No" onClick="label.value = self.label"/>
</window>
```

### UUID

A component has another identifier called UUID (Universal Unique ID), which application developers rarely need.

UUID is used by components and Client Engine to manipulate DOM at the browser and to communicate with the server. More precisely, the id attribute of a DOM element at the client is UUID.

UUID is generated automatically when a component is created. It is immutable, except the identifiers of components for representing HTML tags.

HTML relevant components handle UUID different from other set of components: UUID is the same as ID. If you change ID of a HTML relevant component, UUID will be changed accordingly. Therefore, old JavaScript codes and servlets will remain to work without any modification.

17 If the client is a browser, the visual representation is a DOM element or a set of DOM elements.

18 The visual part is created, updated and removed automatically. Application developers rarely need to notice its existence. Rather, they manipulate the object part in the server.

## The ID Space

It is common to decompose a visual representation into several ZUML pages. For example, a page for displaying a purchase order, and a modal dialog for entering the payment term. If all components are uniquely identifiable in the same desktop, developers have to maintain the uniqueness of all identifiers for all pages that might created to the same desktop.

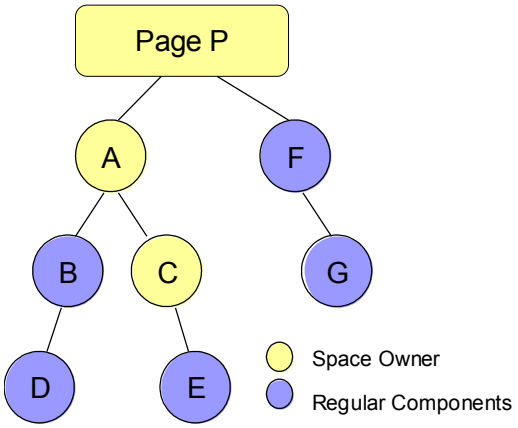
The concept of ID spaces is then introduced to resolved this issue. An ID space is a subset of components of a desktop. The uniqueness is guaranteed only in the scope of an ID space.

The simplest form of an ID space is a window (`com.potix.zul.html.Window`). All descendant components of a window (including the window itself) forms an independent ID space. Thus, you could use a window as the topmost component of each page, such that developers need to maintain the uniqueness of each page separately.

More generally, any component could form an ID space as long as it implements the `com.potix.zk.ui.IdSpace` interface. Page also implements the `IdSpace` interface, so it is also a space owner.

The topmost component of an ID space is called the owner of the ID space, which could be retrieved by the `getSpaceOwner` method in the `Component` interface.

If an ID space, say X, is a descendant of another ID space, say Y, then space X's owner is part of space Y but descendants of X is not part of space Y.




As depicted in the figure, there are three spaces: P, A and C. Space P includes P, A, F and G. Space A includes A, B, C and D. Space C includes C and E.

Components in the same ID spaces are called fellows. For example, A, B, C and D are fellows of the same ID space.

To retrieve another fellow, you could use the `getFellow` method in the `IdSpace` interface or the `Component` interface.

Notice that the `getFellow` method can be invoked against any components in the same ID space, not just the space owner. Similarly, the `getSpaceOwner` method returns the same object for any components in the same ID space, no matter it is the space owner or not.

The `com.potix.zk.ui.Path` class provides utilities to simplify the location of a component among ID spaces. Its use is similar to `java.io.File`.

```
Path.getComponent("/A/C/E");
new Path("A/C", "E").getComponent();
```

### Variable and Functions Defined in `zscript`

In addition to executing codes, you could define variables and functions in the `zscript` element as depicted below.

```
<window id="A>
   <zscript>
      Object myvar = new LinkedList();
      void myfunc() {
         ...
      }
   </zscript>
   ...
   <button label="add" onClick="myvar.add(some)"/>
   <button label="some" onClick="myfunc()"/>
</window>
```

The variables and functions defined in `zscript` are stored in the ID space that contains it. In the above case, both `myvar` and `myfunc` are stored in the ID space of window A.

> **Tip:** Components assigned with ID are actually defined as variables of the ID space it belongs.

A child ID space can see all variables and functions defined in its parent, but not vice versa. Thus, in the follow example, window A sees only `var1`, while window B sees both `var1` and `var2`.

```
<window id="A">
   <zscript>var1 = "abc";</zscript>
   <window id="B">
      <zscript>var2 = "def";</zscript>
   </window>
</window>
```

It is interesting to note that, when an assignment occurs, ZK will locate the variable first. If found, it updated it directly. If not found, it creates a new variable at the ID space where `zscript` is evaluated. Thus, in the following example, `var2` is "def". In other words, the second `var1` is the same as the first one.

```
<window id="A">
   <zscript>var1 = "abc";</zscript>
   <window id="B">
      <zscript>var1 = "def";</zscript>
   </window>
   <zscript>var2 = var1;</zscript>
</window>
```

To declare a variable local to a namespace no matter whether its parent ID space defined it or not, use the following format:

MyClass myvar = myvalueThus, in the following example, var2 is "abc". In other words, the

second `var1` is created in the ID space of window B, while the first `var1` is created in the ID space of window A.

```
<window id="A">
   <zscript>var1 = "abc";</zscript>
   <window id="B">
      <zscript>Object var1 = "def";</zscript>
   </window>
   <zscript>var2 = var1;</zscript>
</window>
```

> **Tip:** Currently, only the window components implement `com.potix.zk.ui.IdSpace`. In other words, only windows and pages are the owners of ID spaces.

### `zscript` and `EL` Expressions

A variable defined in zscript is accessible by EL expressions, as long as they are in the same ID space.

```
<window>
   <zscript>
   String var = "abc";
   </zscript>
   ${var}
</window>
```

is equivalent to

```
<window>
   abc
</window>
```

To declare a variable that is not visible to the ID space (including EL expressions), you have to declare it as a local variable by specify the class name and enclose it with curly braces as follows.

```
<zscript>
{
   String var = "abc"; //visible only inside of the curly brace
}
</zscript>
```

Then, it is not visible to the ID space and EL expressions it contains.

## Events

An event (`com.potix.zk.ui.event.Event`) is used to notify application what happens. Each type of event is represented by a distinct class. For example, `com.potix.zk.ui.event.MouseEvent` denotes a mouse activity, such as clicking.

To response to an event, an application must register one or more event listeners to it. There are two ways to register an event listener. One is by specifying the `onXxx` attribute in the markup language. The other is by calling the `addEventListener` method for the component or the page you want to listen.

In addition to event triggered by user's activity at the browser, an application could fire events by use of the `sendEvent` and `postEvent` methods from the `com.potix.zk.ui.event.Events` class.

## Desktops and Event Processing

As mentioned above, a desktop is a collection of pages for serving the same URL request. A desktop is also the scope that an event listener could access.

When an event is fired, it is associate with a desktop. ZK separates events based on the associated desktops, and pipelines events into separated queues. Therefore, events for the same desktop are processed sequentially. On the other hand, events for different desktops are processed in parallel.

An event listener is allowed to access any components of any pages of the desktop associated with the event. It is also allowed to moving components from one page to another as long as they are in the same desktop. On the other hand, it *cannot* access components belonging to other desktops. Because the binding of a component and a desktop is permanent, developers *cannot* detach a component from one desktop, and then reuse it in another desktop.

### Desktops and the Creation of Components

When a component is created in an event listener, it is assigned automatically to the associated desktop of the event being processed. This assignment happens even if the component is *not* attached to a page. It means that any component you created in an event listener can be used in the same desktop that the listener is handling.

When a component is created in a thread other than any event listener, it doesn't belong to any desktop. In this case, you could attach to any desktop you want as long as the attachment occurs in a proper event listener. Of course, once the component is attached to a desktop, it belongs to the desktop forever.

For most applications, it is rarely necessary to create components in a thread other than event listeners. However, if you have a long operation, you might want to execute it in a background thread. Then, you could prepare the component tree at the background and then add it to a desktop when a proper event is received. Refer to the **Long Operations** section in the **Event Listening and Processing** chapter.

## ZUML and XML Namespaces

The ZK User Interface Markup Language (ZUML) is a XML-based language used by developers to describe the visual representation. ZUML is aimed to separate the dependency of the set of components to use. In other words, different set of components[19], such as XUL and XHTML, could be used simultaneously in the same ZUML page. Different markup languages could be added transparently. If two or more set of components are used in the same page, developers have to use the XML namespaces to distinguish them. Refer to the **Component Sets and XML Namespaces** section in the **ZK User Interface Markup Language** chapter if you want to mix multiple component sets, say XUL and XHTML, in the same page.

> **Tip:** Using XML namespaces in ZUML is optional. You need it only if you want to mix two or more.

19 Also known as tags. There is one-to-one mapping between components and tags.

# 4. The Component Lifecycle

This chapter describes the lifecycles of loading pages and updating pages.

## The Lifecycle of Loading Pages

It takes four phases for ZK loaders to load and interpret a ZUML page: the Page Initial Phase, the Component Creation Phase, the Event Processing Phase, and the Rendering Phase.

### The Page Initial Phase

In this phase, ZK processes the processing instructions, called `init`. If none of such processing instructions are defined, this phase is skipped.

For each `init` processing instruction with the `class` attribute, an instance of the specified class is constructed, and then its `doInit` method is called. What the class will do, of course, depends on your application requirements.

```
<?init class="MyInit"?>
```

Another form of the `init` processing instruction is to specify a `zscript` file as follows. Then, the `zscript` file will be interpreted at the Phase Initial phase.

```
<?init zscript="/my/init.zs"?>
```

Notice that the page is not yet attached to the desktop when the Page Initial phase executes.

### The Component Creation Phase

In this phase, ZK loader interprets an ZUML page. It creates and initializes components accordingly. It takes several steps as follows.

1. For each element, it examines the `if` and `unless` attribute to decide whether it is effective. If not, the element and all of its child elements are ignored.
2. If the `forEach` attribute is specified with a collection of items, ZK repeats the following steps for each item in the collection.
3. Creates a component based on the element name, or by use of the class specified in the `use` attribute, if any.
4. Initializes the members one-by-one based on the order that attributes are specified in the ZUML page.
5. Interprets the nested elements and repeat the whole procedure.

6. After all children are created, the `onCreate` event is sent to this component, such that application could initialize the content of some elements later. Notice that the `onCreate` events are posted for child components first.

#### The Event Processing Phase

In this phase, ZK invokes each listener for each event queued for this desktop one-by-one.

An independent thread is started to invoke each listener, so it could be suspended without affecting the processing of other events.

During the processing, an event listener might fire other events. Refer to the **Event Listening and Processing** chapter for details.

#### The Rendering Phase

After all events are processed, ZK renders these components into a regular HTML page and sends this page to the browser.

To render a component, the `redraw` method is called. The implementation of a component shall not alter any content of the component in this method.

## The Lifecycle of Updating Pages

It takes three phases for ZK AU Engine to process the ZK requests sent from the clients: the Request Processing Phase, the Event Processing Phase, and the Rendering Phase.

ZK AU Engine pipelines ZK requests into queues on a basis of one queue per desktop. Therefore, requests for the same desktop are processed sequentially. Requests for different desktops are processed in parallel.

#### The Request Processing Phase

Depending on the request, ZK AU Engine might update the content of affected components such that their content are the same as what are shown at the client.

Then, it posts corresponding events to the queue.

#### The Event Processing Phase

This phase is the same as the Event Processing Phase in the Component Creation Phase. It processes events one-by-one in an independent thread.

#### The Rendering Phase

After all events are processed, ZK renders affected components, generates corresponding ZK

responses, and sends these responses back to the client. Then, Client Engine updates the DOM tree at the browser based on the responses.

Whether to redraw the whole visual representation of a component or to update an attribute at the browser all depend on the implementation of components. It is the job of component developers to balance between interactivity and simplicity.

## The Molds

A component could have different appearance even at the same page. The concept is called mold (aka., template). Developers could dynamically change the mold by use of the `setMold` method in the `Component` interface. All components support a mold called `default`, which is the default value. Some components might have support two or more molds For example, `tabbox` supports both `default` and `accordion` molds.

```
<tabbox><!-- if not specified, the default mold is assumed. -->
   <tabs>
      <tab label="Default"/>
   </tabs>
   <tabpanels>
      <tabpanel>
      <tabbox mold="accordion">
         <tabs>
            <tab label="First Accordion"/>
            <tab label="Second Accordion"/>
         </tabs>
         <tabpanels>
            <tabpanel>The first panel.</tabpanel>
            <tabpanel>The second panel.</tabpanel>
         </tabpanels>
      </tabbox>
      </tabpanel>
   </tabpanels>
</tabbox>
```


## Component Garbage Collection

Unlike many component-based GUI, ZK has no destroy or close method for components. Like W3C DOM, a component is removed from the browser as soon as it is detached from the page. It is shown as soon as it is attached to the page.

More precisely, once a component is detached from a page, it is no longer managed by ZK. If the application doesn't have any reference to it. The memory occupied by the component will be released by JVM's Garbage Collector.

# 5. Event Listening and Processing

This chapter describes how an event is processed.

## Add Event Listeners by Markup Languages

The simplest way to add an event listener is to declare an attribute in a ZUML page. The value of the attribute for listening an event is any Java codes that could be interpreted by BeanShell.

```
<window title="Hello" border="normal">
   <button label="Say Hello" onClick="alert(&quot;Hello World!&quot;)"/>
</window>
```

## Add and Remove Event Listeners by Program

There are two ways to add event listeners by program.

### Declare a Member

When overriding a component by use of your own class, you could declare a member function to be an event listener as follows.

In a ZUML page, you declare the `use` attribute to specify what class you want to use instead of the default one. As illustrated below, it asks ZK to use the MyClass class instead of `com.potix.zul.html.Window`[20].

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

Then, you implement MyWindow.java by extending from the default class as follows.

```
public class MyWindow extends com.potix.zul.html.Window {
   public void onOK() { //add an event listener
       ...//handles the onOK event (sent when ENTER is pressed)
   }
}
```

If you want to retrieve more information about the event, you could declare as follows.

```
public void onOK(com.potix.zk.ui.event.KeyEvent event) {
...
}
```

Different events might be associated with different event objects. Refer to **Append C** for

20 The default class is defined in lang.xml embedded in zul.jar.

more details.

### Add and Remove Event Listeners Dynamically

Developers could use the `addEventListener` and `removeEventListener` methods of the `com.potix.zk.ui.Component` interface to dynamically add or remove an event listener. As illustrated below, the event listener to be added dynamically must implement the `com.potix.zk.ui.event.EventListener` interface.

```
void init(Component comp) {
   ...
   comp.addEventListener("onClick", new MyListener());
   ...
}
class MyListener implements com.potix.zk.ui.event.EventListener {
   public void onEvent(Event event) throws UiException {
      ...//processing the event
   }
   public boolean isAsap() {
      return true; //Refer the following section for description
   }
}
```

### What ASAP Is?

The `isAsap` method in the `EventListener` interface defines the emergency of the listener. If it returns true[21], the event will be sent from the browser to the server as soon as it happens.

By returning false, the event won't be sent until another ASAP event is about to sent. The performance of the server is then improved, because the communication frequency between client and server is reduced.

Notice that it won't affect the correctness, because an application remains idle until an event is received and the order of arriving events are the same.

### Add and Remove Event Listeners to Pages Dynamically

Developers could add event listeners to a page (`com.potix.zk.ui.Page`) dynamically. Once added, all events of the specified name the are sent to any components of the specified page will be sent to the listener.

All page-level event listeners are non-ASAP. In other words, the `isArap` method is ignored.

A typical example is to use a page-level event listener to maintain the modification flag as follows.

```
public class ModificationListener implements EventListener {
   private final Window _owner;
```

21 ASAP stands for As Soon As Possible.

```
   private final Page _page;
   private boolean _modified;

   public ModificationListener(Window owner) {
      //Note: we have to remember the page because unregister might
      //be called after the owner is detached
      _owner = owner;
      _page = owner.getPage();
      _page.addEventListener("onChange", this);
      _page.addEventListener("onSelect", this);
      _page.addEventListener("onCheck", this);
   }
   /** Called to unregister the event listener.
    */
   public void unregister() {
      _page.removeEventListener("onChange", this);
      _page.removeEventListener("onSelect", this);
      _page.removeEventListener("onCheck", this);
   }
   /** Returns whether the modified flag is set.
    */
   public boolean isModified() {
      return _modified;
   }
   //-- EventListener --//
   public void onEvent(Event event) throws UiException {
      _modified = true;
   }
   public boolean isAsap() {
      return false;
   }
}
```

### The Invocation Sequence

The sequence of invoking event listeners is as follows. Let us assume the `onClick` event is received.

1. Invoke the script specified in the `onClick` attribute of the targeting component, if any.
2. Invoke event listeners for the `onClick` event one-by-one that are added to the targeting component. The first added, the first called.
3. Invoke the `onClick` member method of the targeting component, if any.
4. Invoke event listeners for the `onClick` event one-by-one that are added to the page that the targeting component belongs. The first added, the first called.

### Abort the Invocation Sequence

You could abort the calling sequence by calling the `stopPropagation` method in the

`com.potix.zk.ui.event.Event` class. Once one of the event listeners invokes this method, all following event listeners are ignored.

## Send and Post Events from an Event Listener

In addition to receiving events, an application could communicate among event listeners by posting or sending events to them.

### Post Events

By use of the `postEvent` method in the `com.potix.zk.ui.event.Events` class, an event listener could post an event to the end of the event queue. It returns immediately after placing the event into the queue. The event will be processed later after all events preceding it have been processed.

### Send Events

By use of the `sendEvent` method in the `com.potix.zk.ui.event.Events` class, an event listener could ask ZK to process the specified event immediately. It won't return until all event listeners of the specified event has been processed. The event is processed at the same thread.

## Thread Model

For each desktop, events are processed sequentially, so thread model is simple. Like developing desktop applications, you don't need to worry about racing and multi-threading. All you need to do is to register an event listener and process the event when invoked.

> **Tip:** Each event listener executes in an independent thread called event processing thread, while the page is evaluated in the servlet thread.

### Suspend and Resume

For advanced applications, you might have to suspend an execution until some condition is satisfied. The `wait`, `notify` and `notifyAll` methods of the `com.potix.zk.ui.Executions` class are designed for such purpose.

When an event listener want to suspend itself, it could invoke `wait`. Another thread could then wake it up by use of `notify` or `notifyAll`, if the application-specific condition is satisfied. The modal dialog is an typical example of using this mechanism.

Their use is similar to the `wait`, `notify` and `notifyAll` methods of the `java.lang.Object` class. However, you cannot use the methods of `java.lang.Object` for suspending and

resuming event listeners. Otherwise, all event processing will be stalled for the associated desktop.

```
public void myEventHandler() {
   ...
   Executions.wait(flag); //suspend until flag is notified
   ... //executes only after resumed
}
public void anotherHandler() {
   ...
   Executions.notifyAll(flag); //resume all thread waiting for flag
   ...
}
```

### Long Operations

Events for the same desktop are processed sequentially. In other words, an event handler will block any following handlers. The time blocking user's requests might not be acceptable, if an event handler takes too much time to execute. Like desktop applications, you have to create a working thread for long operations to minimize the blocking time.

Due to the limitations of HTTP, you have to conform with the following rules.

- Use the `wait` method in the `com.potix.zk.ui.Executions` class to suspend the event handler itself, after creating a working thread.
- Because the working thread is not an event listener, it *cannot* access any components, unless the components don't belong to any desktop. Thus, you might have to pass necessary information manually before starting the working thread.
- Then, the working thread could crush the information and create components as necessary. Just don't reference any component that belongs to any desktop.
- Use the `notify(Page page, Object flag)` or `notifyAll(Page page, Object flag)` method in the `com.potix.zk.ui.Executions` class in the working thread to resume the event handler, after the working thread finishes.
- The resumed event handler won't be executed immediately until another event is sent from the client. To enforce an event to be sent, you could use a timer component (`com.potix.zul.html.Timer`) to fire an event a moment later or periodically. This event listener for this timer could do nothing or update the progress status.

```
public void myEventHandler() {
  ... //retrieve info from components for the working thread
  worker = new WorkingThread(info);
  worker.start();
  Executions.wait(flag); //wait for working thread to complete
  worker.listbox.setParent(main);
    //attach listbox that is prepared by the working thread
  ...
```

```
}
class WorkinigThread extends Thread {
  public Listbox listbox;
  public boolean done;
  public void run() {
    ...//does the lengthy operation
    listbox = new Listbox(); //it doesn't belongs to any desktop
    for (...) {
      ...
      new Listitem(something).setParent(l);
    }
    //listbox is ready to attach
    done = true;
    Executions.notify(page, flag); //resume the event handler
  }
}
```

And, a ZUML page to control a timer and to display the progress might be as follows.

```
<window>
  <label id="progress"/>
  <timer id="tm" delay="1000">
    <attriute name="onTimer">
   if (worker.done)
     tm.stop(); //stop the timer
   else
     progress.value = percentage; //show the percentag
    </attribute>
  </timer>
</window>
```

## Initialization and Cleanup of Event Processing Thread

### Initialization Before Processing Each Event

An event listener is executed in an event processing thread. Sometimes, you have to initialize the thread before processing any event.

A typical example is to initialize the thread for the authentication. Some J2EE or Web containers store authentication information in the thread local storage, such that they could re-authenticate automatically when needed.

To initialize the event processing threads, you have to register a class, that implements the `com.potix.zk.ui.event.EventThreadInit` interface, to the `listener` element in the `WEB-INF/zk.xml` file[22].

Once registered, an instance of the specified class is constructed in the main thread (aka., the servlet thread), before starting an event processing thread. Then, the `init` method of

22 It is described more detailedly in **Appendix B**.

the instance is called at the context of the event processing thread before doing anything else.

Notice that the constructor and the `init` method are invoked at different thread such that developers could retrieve thread-dependent data from one thread and pass to anther.

Here is an example for the authentication mechanism of JBoss[23]. In this example, we retrieve the information stored in the servlet thread in the constructor. Then, we initialize the event processing thread when the `init` method is called.

```
import java.security.Principal;
import org.jboss.security.SecurityAssociation;
import com.potix.zk.ui.Component;
import com.potix.zk.ui.event.Event;
import com.potix.zk.ui.event.EventThreadInit;

public class JBossEventThreadInit implements EventThreadInit {
   private final Principal _principal;
   private final Object _credential;
   /** Retrieve info at the constructor, which runs at the servlet thread. */
   public JBossEventThreadInit() {
      _principal = SecurityAssociation.getPrincipal();
      _credential = SecurityAssociation.getCredential();
   }
   //-- EventThreadInit --//
   /** Initial the event processing thread at this method. */
   public void init(Component comp, Event evt) {
      SecurityAssociation.setPrincipal(_principal);
      SecurityAssociation.setCredential(_credential);
   }
}
```

Then, in `WEB-INF/zk.xml`, you have to specify as follows.

```
<zk>
   <listener>
      <listener-class>JBossEventThreadInit</listener-class>
   </listener>
</zk>
```

### Cleanup After Processed Each Event

Similarly, you might have to clean up an event processing thread after it has processed an event.

A typical example is to close the transaction, if it is not closed properly.

To cleanup the event processing threads, you have to register a listener class, that implements the `com.potix.zk.ui.event.EventThreadCleanup` interface, to the `listener` element in the `WEB-INF/zk.xml` file.

23 http://www.jboss.org

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

# 6. The ZK User Interface Markup Language

The ZK User Interface Markup Language (ZUML) is based on XML. Each XML element describes what component to create. A XML attribute describes an initial values to be assigned to the created component. An XML processing instruction describes how to process the whole page, such as the page title.

Different sets of components are distinguished by XML namespaces. For example, the namespace of XUL is `http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul`, and that of XHTML is `http://www.w3.org/1999/xhtml`.

## XML

This section provides the most basic concepts of XML to work with ZK. If you are familiar with XML, you could skip this section. If you want to learn more, there are a lot of resources on Internet, such as http://www.w3schools.com/xml/xml_whatis.asp and http://www.xml.com/pub/a/98/10/guide0.html.

XML is a markup language much like HTML but with stricter and cleaner syntax. It has several characteristics worth to notice.

### Elements Must Be Well-formed

First, each element must be closed. They are two ways to close an element as depicted below. They are equivalent.

| | |
|---|---|
| Close by an end tag: | `<window></window>` |
| Close without an end tag: | `<window/>` |

Second, elements must be properly nested.

| | |
|---|---|
| Correct: | `<window>`<br>`  <groupbox>`<br>`  Hello World!`<br>`  </groupbox>`<br>`</window>` |
| Wrong: | `<window>`<br>`  <groupbox>`<br>`  Hello World!`<br>`  </window>`<br>`</groupbox>` |

### Special Character Must Be Replaced

XML use <*element-name*> to denote an element, so you have to replace special characters. For example, you have to use `<` to represent the < character.

| Special Character | Replaced With |
|---|---|
| < | < |
| > | > |
| & | & |
| " | " |
| ' | ' |

Alternatively, you could ask XML parser not to interpret a piece of text by use of `CDATA` as follows.

```
<zscript>
<![CDATA[
void myfunc(int a, int b) {
   if (a < 0 && b > 0) {
      //do something
   }
]]>
</script>
```

It is interesting to notice that backslash (\) is not a special character, so you don't need to escape it at all.

### Attribute Values Must Be Specified and Quoted

| | |
|---|---|
| Correct: | `width="100%"`<br>`checked="true"` |
| Wrong: | `width=100%`<br>`checked` |

### Comments

A comment is used to leave a note or to temporarily edit out a portion of XML code. To add a comment to XML, use <!-- and --> to escape them.

```
<window>
<!-- this is a comment and ignored by ZK -->
</window>
```

### Character Encoding

It is, though optional, a good idea to specify the encoding in your XML such that the XML parser can interprets it correctly. Note: it must be the first line of the file.

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

In addition to specify the correct encoding, you have to make sure your XML editor supports it as well.

### Namespace

Namespaces are a simple and straightforward way to distinguish names used in XML documents. ZK uses XML namespaces to distinguish the component name, such that it is OK to have two components with the same name as long as they are in different namespace. In other words, ZK uses a XML namespace to represent a component set, such that developers could mix two or more component sets in the same page, as depicted below.

```
<html xmlns:="http://www.w3.org/1999/xhtml"
 xmlns:x="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
 xmlns:zk="http://www.potix.com/2005/zk">
<head>
<title>ZHTML Demo</title>
</head>
<body>
   <h1>ZHTML Demo</h1>
   <table>
   <tr>
      <td><x:textbox/></td>
      <td><x:button label="Now" zk:onClick="addItem()"/></td>
   </tr>
   </table>

   <zk:zscript>
   void addItem() {
   }
   </zk:zscript>
</body>
</html>
```

where

- `xmlns:x="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"` specifies a namespace called `http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul`, and use x to represent this namespace.
- `xmlns:="http://www.w3.org/1999/xhtml"` specifies a namespace called `http://www.w3.org/1999/xhtml`, and use it as the default namespace.
- `<html>` specifies an element called `html` from the default namespace, i.e., `http://www.w3.org/1999/xhtml` in this example.
- `<x:textbox/>` specifies an element called `textbox` from the name space called `http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul`.

## Conditional Evaluation

The evaluation of an element could be conditional. By specifying the `if` or `unless` attribute or both, developers could control whether to evaluate the associated element.

In the following example, the window component is created only if a is 1 and b is not 2. If an element is ignored, all of its child elements are ignored, too.

```
<window if="${a==1}" unless="${b==2}">
   ...
</window>
```

The following example controls when to interpret a piece of Java codes.

```
<textbox id="contributor"/>
<zscript if="${param.contributor}">
   contributor.label = Executions.getCurrent().getParameter("contributor");
</zscript>
```

## Iterative Evaluation

The evaluation of an element could be iterative. By specifying a collection of objects to the `forEach` Attribute, developers could control how many time of the associated element shall be evaluated. For sake of description, we call an element is an iterative element if it is assigned with the `forEach` attribute.

In the following example, the list item is created three times. Notice that you have to use EL expression to specify the collection.

```
<listbox>
  <zscript>
  grades = new String[] {"Best", "Better", "Good"};
  </zscript>
  <listitem label="${each}" forEach="${grades}"/>
</listbox>
```

The iteration depends on the type of the specified value of the `forEach` attribute.

- If `java.util.Collection`, it iterates each element of the collection.
- If `java.util.Map`, it iterates each `Map.Entry` of the map.
- If `java.util.Iterator`, it iterates each element from the iterator.
- If `java.util.Enumeration`, it iterates each element from the enumeration.
- If `Object[]`, `int[]`, `short[]`, `byte[]`, `char[]`, `float[]` or `double[]` is specified, it iterates each element from the array.
- If null, nothing is generated (it is ignored).

- If neither of above types is specified, the associated element will be evaluated once as if a collection with a single item is specified.

```
<listbox>
  <listitem label="${each}" forEach="grades"/>
</listbox>
```

### The `each` Variable

During the evaluation, a variable called `each` is created and assigned with the item from the specified collection. In the above example, `each` is assigned with "Best" in the first iteration, then "Better" and finally "Good".

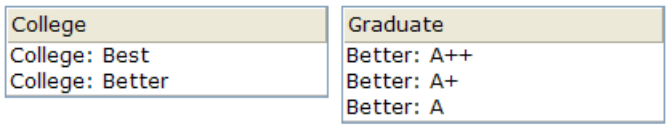
Notice that the `each` variable is accessible both in EL expression and in `zscript`. ZK will preserve the value of the `each` variable if it is defined before, and restore it after the evaluation of the associated element.

### The `forEachStatus` Variable

The `forEachStatus` variable is an instance of `com.potix.ui.util.ForEachStatus`. It holds the information about the current iteration. It is mainly used to get the item of the enclosing element that is also assigned with the `forEach` attribute.

In the following example, we use nested iterative elements to generate two listboxes.

```
<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>
```




Notice that the `forEachStatus` variable is accessible both in EL expression and in `zscript`.

### How to Use `each` and `forEachStatus` Variables in Event Listeners

It is a bit tricky to use the `forEach` and `forEachStatus` variables in event listeners, because they are available only in the Component Creation Phase[24]. Thus, the following sample is *incorrect*: when the `onClick` listener is called, the each variable is no longer available.

```
<window title="Countries" border="normal" width="100%">
   <zscript><![CDATA[
   String[] countries = {
      "China", "France", "Germany", "United Kindom", "United States"};
   ]]></zscript>

   <hbox>
      <button label="${each}" forEach="${countries}"
         onClick="alert(each)"/> <!-- incorrect!! -->
   </hbox>
</window>
```

Notice that the button's label is assigned correctly because it is done at the same phase – the Component Creation Phase.

Also notice that you cannot use EL expressions in the event listener. For example, the following codes fail to execute because the `onClick` listener is not a legal Java codes (i.e., EL expressions are ignored in `zscript`).

```
<button label="${each}" forEach="${countries}"
      onClick="alert(${each})"/> <!-- incorrect!! -->
```

#### A Solution: `custom-attributes`

The solution is that we have to store the content of `each` (and `forEachStatus`) somewhere such that its content is still available when the listener executes. You can store its content anywhere, but there is a simple way to do it as follows.

```
<window title="Countries" border="normal" width="100%">
   <zscript><![CDATA[
   String[] countries = {
      "China", "France", "Germany", "United Kindom", "United States"};
   ]]></zscript>

   <hbox>
      <button label="${each}" forEach="${countries}"
      onClick="alert(self.getAttribute(&amp;quot;country&amp;quot;))">
         <custom-attributes country="${each}"/>
      </button>
   </hbox>
</window>
```

Like button's label, the properties of custom attributes are evaluated in the Component

24 Refer to the **Component Lifecycle** chapter for more details.

Creation Phase, so you can use `each` there. Then, it is stored to a custom attribute which will last as long as the component exists (or until being removed programmingly).

## Implicit Objects

For scripts embedded in a ZUML page, there are a set of implicit objects that enable developers to access components more efficiently. These objects are available to the Java codes included by the `zscript` element and the attributes for specifying event listeners. They are also available to EL expressions.

For example, `self` is an instance of `com.potix.zk.ui.Component` to represent the component being processing. In the following example, you could identify the component in an event listener by `self`.

```
<button label="Try" onClick="alert(self.label)"/>
```

Similarly, `event` is the current event being processed by an event listener. Thus, the above statement is equivalent to

```
<button label="Try" onClick="alert(event.target.label)"/>
```

**List of Implicit Objects**

| Object Name | Description |
|---|---|
| `self` | `com.potix.zk.ui.Component`<br>The component itself. |
| `spaceOwner` | `com.potix.zk.ui.IdSpace`<br>The space owner of this component. It is the same as `self.spaceOwner`. |
| `page` | `com.potix.zk.ui.Page`<br>The page. It is the same as `self.page`. |
| `desktop` | `com.potix.zk.ui.Desktop`<br>The desktop. It is the same as `self.desktop`. |
| `session` | `com.potix.zk.ui.Session`<br>The session. It is similar to `javax.servlet.http.HttpSession`[25]. |

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

| Object Name | Description |
| --- | --- |
| componentScope | java.util.Map<br>A map of attributes defined in the component. It is the same as the getAttributes method in the com.potix.zk.ui.Component interface. |
| spaceScope | java.util.Map<br>A map of attributes defined in the ID space containing this component. |
| pageScope | java.util.Map<br>A map of attributes defined in the page. It is the same as the getAttributes method in the com.potix.zk.ui.Page interface. |
| desktopScope | java.util.Map<br>A map of attributes defined in the desktop. It is the same as the getAttributes method in the com.potix.zk.ui.Desktop interface. |
| sessionScope | java.util.Map<br>A map of attributes defined in the session. It is the same as the getAttributes method in the com.potix.zk.ui.Session interface. |
| applicationScope | java.util.Map<br>A map of attributes defined in the web application. It is the same as the getAttributes method in the com.potix.zk.ui.WebApp interface. |
| arg | java.util.Map<br>The arg argument passed to the createComponents method in the com.potix.zk.ui.Executions class. It might be null, depending on how createComponents is called.<br>It is the same as self.desktop.execution.arg. |
| each | java.lang.Object<br>The current item of the collection being iterated, when ZK evaluates an iterative element. An iterative element is an element with the forEach attribute. |
| forEachStatus | com.potix.zk.ui.util.ForEachStatus<br>The status of an iteration. ZK exposes the information relative to the iteration taking place when evaluating the iterative element. |

| Object Name | Description |
|---|---|
| event | com.potix.zk.ui.event.Event or derived<br>The current event. Available for the event listener only. |

#### Information about Request and Execution

The com.potix.zk.ui.Execution interface provides information about the current execution, such as the request parameters. To get the current execution, you could do one of follows.

- If you are in a component, use `getDesktop().getExecution()`.
- If you don't have any reference to component, page or desktop, use the `getCurrent` method in the `com.potix.zk.ui.Executions` class.

## Processing Instructions

The XML processing instructions describe how to process the ZUML page.

#### The `page` Directive

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

It describes attributes of a page.

| Attribute Name | Description |
|---|---|
| id | [Optional][Default: *generated automatically*]<br>Specifies the identifier of the page, such that we can retrieve it back.<br>Refer to the **Identify Pages** section in the **Advanced Features** chapter for details. |
| title | [Optional][Default: *none*]<br>Specifies the page title that will be shown as the title of the browser.<br>It can be changed dynamically by calling the `setTitle` method in the `com.potix.zk.ui.Page` interface. |
| style | [Optional][Default: `width:100%`]<br>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. |

| Attribute Name | Description |
|---|---|
| language | [Optional][Default: *depending on the extension*][xul/html \| xhtml]<br>Specifies the language of this page.<br>Currently, it supports xul/html and xhtml. |

### The xml-stylesheet Directive

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

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

| Attribute Name | Description |
|---|---|
| href | [Required]<br>A hyper link to the style sheet.<br>Like other URI, it accepts "*" for loading browser and Locale dependent style sheet. Refer to the *Browser and Locale Dependent URI* section in the *Internationalization* chapter for details. |
| type | [Optional][Default: text/css]<br>Specifies the type of the style sheet. Currently, it supports only text/css. |

### The component Directive

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

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

Defines a new component for a particular page. Components defined in this directive is visible only to the page with this directive. To define components that can be used in any page, use the language addon, which is a XML file defining components for all pages in a Web application[26].

There are two formats: by-macro and by-class.

#### The by-macro Format

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

26 Refer to the **Developer's Reference** for details.

Defines a new component based on a ZUML page. It is called *a macro component*. In other words, once an instance of the new component is created, it creates child components based on the specified ZUML page (the `macro-uri` attribute). For more details, refer to the **Macro Components** chapter.

#### The by-class Format

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

Defines a new component, if the `extends` attribute is not specified, based on a class. It is called a *native component*. The class must implement the `com.potix.zk.ui.Component` interface.

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

In addition to defining a brand-new component, you can override properties of existent components by specifying `extends="existentName"`. In other words, if `extends` is specified, the definition of the specified component is loaded as the default value and then override only properties that are specified in this directive.

For example, assume you want to define a new component called `mywindow` by use of `MyWindow` instead of the default window, `com.potix.zul.html.Window` in a ZUML page. Then, you can declare it as follows.

```
<?component name="mywindow" extends="window" class="MyWindow"?>
...
<mywindow>
...
</mywindow>
```

It is equivalent to the following codes.

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

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

```
<?component name="okbutton" extends="button" label="OK"
  style="border:1px solid blue"?>
```

Notice the new component name can be the same as the existent one. In this case, all instances of the specified type of component will use the initial properties you assigned, as if it hides the existent definition. For example, the following codes make all buttons to have a blue border as default.

```
<?button name="button" extends="button" style="border:1px solid blue"?>
<button/> <!-- with blue border -->
```

For more information, refer to the **Developer's Reference**.

| Attribute Name | Description |
|---|---|
| `name` | [Required]<br>The component name. |
| `macro-uri` | [Required, if the by-macro format is used]<br>Used by the by-macro format to specify the URI of the ZUML page, which is used as the template to create components. |
| `class` | [Optional]<br>Used with both the `by-class` and `by-macro` formats to specify the class to instantiate an instance of such kind of components. |
| `extends` | [Optional]<br>Used with the `by-class` format to denote the component name to use its properties as the default value, and then override only properties that are specified in this directive.<br>If not specified, any existent definition is ignored. The new component is brand-new, and defined completely with properties specified in this directive. |
| `mold-name` | [Optional][Default: default]<br>Used with the `by-class` format to specify the mold name. If `mold-name` is specified, `mold-uri` must be specified, too. |
| `mold-uri` | [Optional]<br>Used with the `by-class` format to specify the mold URI. If `mold-uri` is specified but `mold-name` is not specified, the mold name is assumed as `default`. |
| *prop1*, *prop2*... | [Optional]<br>Used with both the `by-class` and `by-macro` formats to specify the initial properties (aka., members) of a component.<br>The initial properties are applied *automatically* if a component is created by ZUML (aka., specified as part of a ZUML page).<br>On the other hand, they are not applied if they are created manually (i.e., by Java codes). If you still want them to be applied, you have to invoke the `applyProperties` method. |

### The `init` Directive

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

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

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

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

You could specify any number of the `init` directive. If you choose the first format, the specified class must implement the `com.potix.zk.ui.util.Initator` interface. Once specified, an instance of the class is constructed and its `doInit` method is called, before the page is evaluated.

In addition, the `doFinally` method is called, after the page has been evaluated. The `doCatch` method is called if an exception occurs. Thus, this directive is not limited to initialization. You could use it for cleanup and error handling.

If you choose the second format, the `zscript` file is evaluated and the arguments (`arg0`, `arg1`,...) will be passed as a variable called `args` whose type is `Object[]`.

| Attribute Name | Description |
|---|---|
| `class` | [Optional]<br>A class name that must implement the `com.potix.zk.ui.util.Initator` interface.<br>The `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. |
| `zscript` | [Optional]<br>A `script` file that will be evaluated in the Page Initial phase. |
| `arg0, arg1, arg2, arg3,...` | [Optional]<br>You could specify any number of arguments. It will be passed to the `doInit` method if the first format is used, or as the `args` variable if the second format is used. Note: the first argument is `arg0`, the second is `arg1` and follows. |

## ZK Attributes

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

### The `use` Attribute

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

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

### The `if` Attribute

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

### The `unless` Attribute

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

### The `forEach` Attribute

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

## ZK Elements

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

### The `zk` Element

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

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

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

```
</window>
```

is equivalent to

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

Then, what is it used for?

#### Multiple Root Elements in a Page

Due to XML's syntax limitation, we can only specify one document root. Thus, if you have multiple root components, you must use `zk` as the document root to group these root components.

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

#### Iteration Over Versatile Components

The `zk` element, like components, supports the `forEach` attribute. Thus, you could use it to generate different type of components depending on the conditions. In the following example, we assume `mycols` is a collection of objects that have several members, `isUseText()`, `isUseDate()` and `isUseCombo()`.

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

| Attribute Name | Description |
|---|---|
| `if` | [Optional][Default: `true`]<br><br>Specifies the condition to evaluate this element. |
| `unless` | [Optional][Default: `false`]<br><br>Specifies the condition *not* to evaluate this element. |

| Attribute Name | Description |
|---|---|
| forEach | [Optional][Default: *ignored*]<br>It specifies a collection of objects, such that the zk element will be evaluated repeatedly against each object in the collection. If not specified or empty, this attribute is ignored. If non-collection object is specified, it is evaluated only once as if a single-element collection is specified. |

### The zscript Element

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

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

| Attribute Name | Description |
|---|---|
| src | [Optional][Default: *none*]<br>Specifies the URI of the file containing Java codes. If specified, the Java codes will be loaded as if they are embedded directly.<br>Note: the file shall contain the Java source codes that can be interpreted by BeanShell. Don't specify a class file (aka. byte codes). |
| if | [Optional][Default: true]<br>Specifies the condition to evaluate this element. |
| unless | [Optional][Default: false]<br>Specifies the condition *not* to evaluate this element. |

### The attribute Element

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.

| Attribute Name | Description |
|---|---|
| name | [Required]<br>Specifies the attribute name. |
| if | [Optional][Default: *none*]<br>Specifies the condition to evaluate this element. |

| Attribute Name | Description |
|---|---|
| unless | [Optional][Default: *none*]<br>Specifies the condition *not* to evaluate this element. |

### The `custom-attributes` element

It defines a set of custom attributes. Custom attributes are objects associated with a particular scope. Acceptable scopes include component, space, page, desktop, session and application.

As depicted below, `custom-attributes` is convenient to assign custom attributes without programming.

```
<window>
   <custom-attributes main.rich="simple" simple="intuitive"/>
</window>
```

It is equivalent to

```
<window>
   <zscript>
      self.setAttribute("main.rich");
   </zscript>
</window>
```

Moreover, you could specify what scope to assign the custom attributes to.

```
<window id="main" title="Welcome">
   <custom-attributes scope="desktop" shared="${main.title}"/>
</window>
```

It is equivalent to

```
<window id="main">
   <zscript>
      desktop.setAttribute("shared", main.title);
   </zscript>
</window>
```

Notice that EL expression is evaluated against the component being created. Sometime it is subtle to notice. For example, `${componentScope.simple}` is evaluated to `null`, in the following codes. Why? It is a shortcut of `<label value="${componentScope.simple}"/>`. In other words, the component, `self`, is the label rather than the window, when the EL is evaluated.

```
<window>
   <custom-attributes simple="intuitive"/>
   ${componentScope.simple}
</window>
```

is equivalent to

```
<window>
   <custom-attributes simple="intuitive"/>
   <label value="${componentScope.simple}"/><!-- self is label not window -->
</window>
```

**Tip:** Don't confuse `<attribute>` with `<custom-attributes>`. They are irrelevant. The `attribute` element is a way to define a XML attribute of the enclosing element, while the `custom-attributes` element is used to assign custom attributes to particular scopes.

| Attribute Name | Description |
|---|---|
| `scope` | [Optional][Default: component]<br>Specifies what scope to associate the custom attributes to. |
| `if` | [Optional][Default: *none*]<br>Specifies the condition to evaluate this element. |
| `unless` | [Optional][Default: *none*]<br>Specifies the condition *not* to evaluate this element. |

## Component Sets and XML Namespaces

To allow mix two or more component sets in the same ZUML page, ZK uses XML namespaces to distinguish different sets of components. For example, the namespace of XUL is `http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul`, and that of XHTML is `http://www.w3.org/1999/xhtml`.

On the other hand, most pages uses only one component set. To make such pages easier to write, ZK determines the default namespace based on the extension. For example, the `xul` and `zul` extensions imply the XUL namespace. Therefore, developers need only to associate ZUML pages with a proper extension, and then don't need to worry about XML namespace any more.

### Standard Namespaces

As stated before, each set of components is associated with an unique namespace. However, developers might develop or use additional components from 3rd party, so here we list only the namespaces that are shipped with the ZK distribution.

| Namespace | Description |
|---|---|
| `http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul` | XUL's namespace. |
| `http://www.w3.org/1999/xhtml` | XHTML's namespace. |
| `http://www.potix.com/2005/zk` | ZK namespace. It is the reserved namespace for |

| Namespace | Description |
|---|---|
| | specifying ZK specific elements and attributes. |

It is optional to specify namespaces in ZUML pages, until there are conflicts. ZK determined which namespace to use by examining the extension of a ZUML page. For the `.zul` and `.xul` extensions, the namespace of XUL is assumed. For `html`, `xhtml` and `zhtml`, the namespace of XHTML is assumed.

To mix with another markup language, you have to use `xmlns` to specify the correct namespace.

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

For the XHTML components, the `onClick` and `onChange` attributes are conflicts with ZK's attributes. To resolve, you have to use the reserved namespace, `http://www.potix.com/2005/zk`, as follows.

```
<?taglib uri="/WEB-INF/tld/zul/core.dsp.tld" prefix="u" ?>

<html xmlns:x="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
xmlns:zk="http://www.potix.com/2005/zk">
<head>
<title>ZHTML Demo</title>
</head>
<body>
   <script>
   function woo() { //running at the browser
   }
   </script>
   <zk:zscript>
   void addItem() { //running at the server
   }
   </zk:zscript>
   <x:window title="HTML App">
     <input type="button" value="Add Item"
      onClick="woo()" zk:onClick="addItem()"/>
   </x:window>
</body>
```

In this example, the `onClick` attribute is a ZHTML's attribute to specify JavaScript codes to run at the browser. On the other hand, the `zk:onClick` is a reserved attribute for specify a ZK event handler.

Notice that the namespace prefix, `zk`, is optional for the `zscript` element, because ZHTML has no such element and ZK has enough information to determine it.

Also notice that you have to specify the XML namespace for the `window` component, because it is from a different component set.

# 7. ZUML with the XUL Component Set

This chapter describes the set of XUL components. Unlike other implementation, XUL components of ZK is optimized for co-operating across Internet. Some components might not be totally compliant with XUL standards. For sake of convenience, we sometimes refer them as ZUL components.

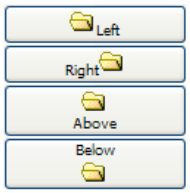
## Basic Components

### Buttons

There are two types of buttons: `button` and `toolbarbutton`. They behave similar except the appearance is different. The `button` component uses HTML BUTTON tag, while the `toolbarbutton` component uses HTML A tag.

You could assign a label and an image to a button by the `label` and `image` attributes. If both are specified, the `dir` attribute control which is displayed up front, and the `orient` attribute controls whether the layout is horizontal or vertical.

```
<button label="Left" image="/img/folder.gif" width="125px"/>
<button label="Right" image="/img/folder.gif" dir="reverse"
width="125px"/>
<button label="Above" image="/img/folder.gif" orient="vertical"
width="125px"/>
<button label="Below" image="/img/folder.gif" orient="vertical"
dir="reverse" width="125px"/>
```




In addition to identifying images by URL, you could assign a dynamically generated image to a button by use of the `setImageContent` method. Refer to the following section for details.

> **Tip:** The `setImageContent` method is supplied by all components that has the `image` attribute. Simplicity put, `setImageContent` is used for dynamically generated images, while `image` is used for images identifiable by URL.

### The `onClick` Event and `href` Attribute

There are two ways to add behaviors to `button` and `toolbarbutton`. First, you could specify a listener for the `onClick` event. Second, you could specify an URL for the `href` attribute. If both are specified, the `href` attribute has the higher priority, i.e., the onClick event won't be sent.

```
<button onClick="do_something_in_Java()"/>
<button href="/another_page.zul"/>
```

#### The `sendRedirect` Method of the `com.potix.zk.ui.Execution` Interface

When processing an event, you could decide to stop processing the current desktop and redirect to anther page by use of the `sendRedirect` method. In other words, the following two buttons are equivalent (from user's viewpoint).

```
<button onClick="Executions.sendRedirect(&quot;another.zul&quot;)"/>
<button href="another.zul"/>
```

Since the onClick event is sent to the server for processing, you could add more logic before invoking `sendRedirect`, such as redirecting to another page only if certain condition is satisfied.

On the other hand, the `href` attribute is processed completely at the client side. Your application won't be noticed, when users clicks on the button.

### Image

An `image` component is used to display an image at the browser. There are two ways to assign an image to an `image` component. First, you could use the `src` attribute to specify a URI where the image is located. This approach is similar to what HTML supports. It is useful if you want to display a static image, or any image that can be identified by URL.

```
<image src="/some/my.jpg"/>
```

#### Locale Dependent Image

Like using any other attributes that accept an URI, you could specify "*" for identifying a Locale dependent image. For example, if you have different image for different Locales, you could use as follows.

```
<image src="/my*.png"
```

Then, assume one of your users is visiting your page with *de_DE* as the preferred Locale. Zk will try to locate the image file called `/my_de_DE.png`. If not found, it will try `/my_de.png` and finally `/my.png`.

Refer to the *Browser and Locale Dependent URI* section in the *Internationalization* chapter for details.

Second, you could use the `setContent` method to assign the content of an image into an `image` component directly. Once assigned, the image displayed at the browser is updated automatically. This approach is useful if an image is generated dynamically.

For example, you could generate a map for the location specified by a user as below.

```
Location: <textbox onChange="updateMap(self.value)"/>
Map: <image id="image"/>
<zscript>
   void updateMap(String location) {
```

```
      if (location.length() > 0)
         image.setContent(new MapImage(location));
   }
</zscript>
```

In the above example, we assume you have a class called `MapImage` for generating a map of the specified location, which is so-called business logic.

Notice that the image component accepts the content only in the `com.potix.image.Image` interface. If the image generated by your tool is not in this format, you could use the `com.potix.image.AImage` class to wrap a binary array of data, a file or an input stream into the `Image` interface.

In traditional Web applications, caching a dynamically generated image is complicate. With the `image` component, you don't need to worry about it. Once the content of an image is assigned, it belongs to the `image` component, and the memory it occupies will be released automatically after the `image` component is no long used.

> **Tip:** If you want to display the contents, say PDF, other than image and audio, you could use the `iframe` component. Refer to the relevant section for details.

### Map

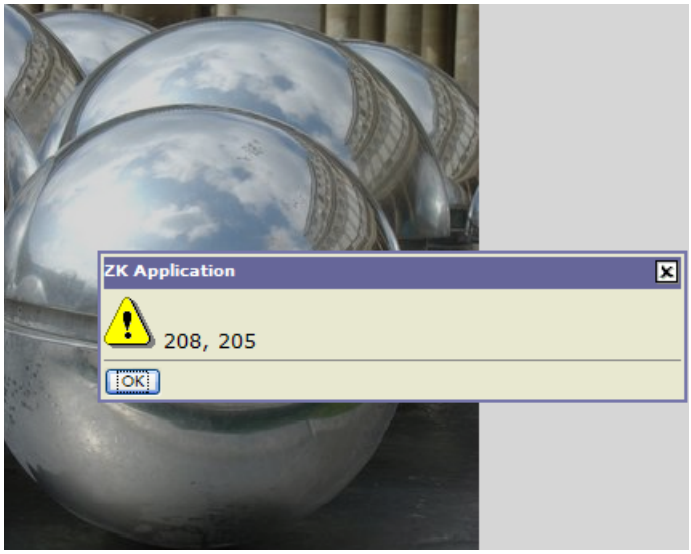
A `map` component is a special image. It accepts whatever attributes an `image` component accepts. However, unlike `image`, if a user clicks on the image, an `onClick` event is sent back to the server with the coordinates of the mouse position. In contrast, the `onClick` event sent by `image` doesn't contain the coordinates.

The coordinates of the mouse position are screen pixels counted from the upper-left corner of the image beginning with (0, 0). It is stored as instance of `com.potix.zk.ui.event.MouseEvent`. Once the application receives the `onClick` event, it could examine the coordinates of the mouse position from the `getX` and `getY` methods.

For example, if a user clicks 208 pixels over and 205 pixels down from the upper-left corner of the image displayed from the following statement.

```
<map src="/img/sun.jpg" onClick="alert(event.x + &quot;, &quot; +event.y)"/>
```

Then, the user gets the result as depicted below.




The application usually uses the coordinates to determine where a user has clicked, and then response accordingly.

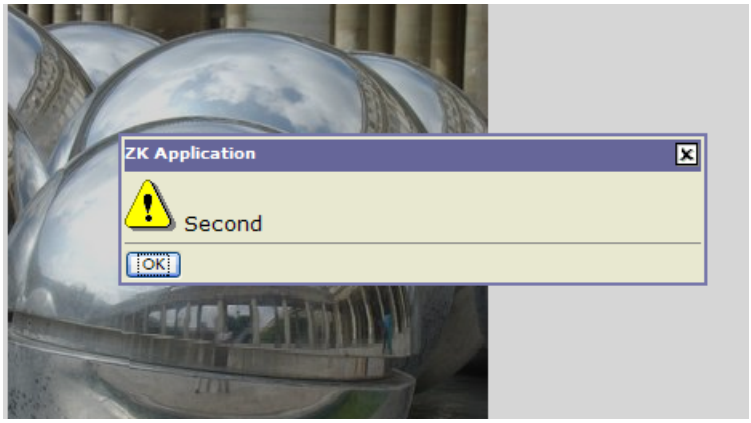
**Area**

Instead of processing the coordinates by the application itself, developers could add the `area` components as the children of a `map` component.

```
<map src="/img/sun.jpg" onClick="alert(event.area)">
  <area id="First" coords="0, 0, 100, 100"/>
  <area id="Second" shape="circle" coords="200, 200, 100"/>
</map>
```

Then, the map component will translate the coordinates of the mouse position to a logical name: the identifier of the area that users has clicked.

For example, if users clicks at (150, 150), then the user gets the result as depicted blow.



### The `shape` attribute

An area component supports three kinds of shapes: circle, polygon and rectangle. The coordinates of the mouse position are screen pixels counted from the upper-left corner of the image beginning with (0, 0).

| Shape | Coordinates / Description |
|---|---|
| `circle` | `coords="x, y, r"`<br>where `x` and `y` define the position of the center of the circle and `r` is its radius in pixels. |
| `polygon` | `coords="x1, y1, x2, y2, x3, y3..."`<br>where each pair of `x` and `y` define a vertex of the polygon. At least thee pairs of coordinates are required to defined a triangle. The polygon is automatically closed, so it is not necessary to repeat the first coordinate at the end of the list to close the region. |
| `rectangle` | `coords="x1, y1, x2, y2"`<br>where the first coordinate pair is one corner of the rectangle and the other pair is the corner diagonally opposite. A rectangle is just a shortened way of specifying a polygon with four vertices. |

If the coordinates in one `area` component overlap with another, the first one takes precedence.

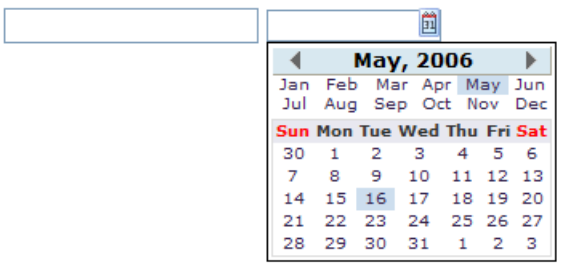
## Audio

An `audio` component is used to play the audio at the browser. Like `image`, you could use the `src` attribute to specify an URL of an audio resource, or the `setContent` method to specify a dynamically generated audio.

Depending on the browser and the audio plugin, developers might be able to control the play of an audio by the `play`, `stop` and `pause` methods. Currently, Internet Explorer with Media Player is capable of such controls.

## Input Controls

A set of input controls are supported in the XUL component set: `textbox`, `intbox`, `decimalbox`, `datebox`, `combobox`, and `bandbox`. They are used to let users input different types of data.




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

```
   <datebox/>
</zk>
```

**Tip:** combobox and bandbox are special input boxes. They shares the common attributes described here. Their unique features will be discussed later in the **Comboboxes** and **Bandboxes** section.

### The `type` attribute

You could specify the `type` attribute with `password` for the `textbox` components, such that what user has entered won't be shown.

```
Username: <textbox/>
Password: <textbox type="password"/>
```

### The `format` Attribute

You could control the format of an input control by the format filed. The default is `null`. For `datebox`, it means `yyyy/MM/dd`. For `intbox` and `decimalbox`, it means no formating at all.

```
<datebox format="MM/dd/yyyy"/>
<decimalbox format="#,##0.##"/>
```

Like any other attributes, you could change the format dynamically, as depicted below.

```
<datebox id="db"/>
<button label="set MM-dd-yyyy" onClick="db.setFormat(&quot;MM-dd-yyyy&quot;)"/>
```

**Mouseless Entry** **datebox**

- `Alt+DOWN` to pop up the calendar.
- `LEFT`, `RIGHT`, `UP` and `DOWN` to change the selected day from the calendar.
- `ENTER` to activate the selection by copying the selected day to the `datebox` control.
- `Alt+UP` or `ESC` to give up the selection and close the calendar.

### Constraints

You could specify what value to accept for input controls by use of the `constraint` attribute. It could a combination of `no positive`, `no negative`, `no zero`, `no empty`, `no future`, `no past`, `no today`, and a regular expression. The first three constraints are applicable only to `intbox` and `decimalbox`. The constraints of `no future`, `no past`, and `no today` are applicable only to `datebox`. The constraint of `no empty` is applicable to any type of components. The constraint of regular expressions is applicable only to String-type input components, such as `textbox`., `combobox` and `bandbox`.

To specify two or more constraints, use comma to separate them as follows.

```
<intbox constraint="no negative,no zero"/>
```

To specify a regular expression, you could have to use / to enclose the regular expression as follows.

```
<textbox constraint="/.+@.+\.[a-z]+/"/>
```

Notes:

- The above statement is XML, so do *not* use \\ to specify a backslash. On the other hand, it is necessary, if writing in Java.

```
new Textbox().setContraint("/.+@.+\\.[a-z]+/");
```

- It is allowed to mix regular expression with other constraints by separating them with comma.

You prefer to display an application dependent message instead of default one, you could append the constraint with colon and the message you want to display when failed.

```
<textbox constraint="/.+@.+\.[a-z]+/: e-mail address only"/>
<datebox constraint="no empty, no future: now or never"/>
```

Notes:

- The error message, if specified, must be the last element and start with colon.
- To support multilingual, you could use the l function as depicted in the **Internationalization** chapter.

```
<textbox constraint="/.+@.+\.[a-z]+/: ${c:l('err.email.required')}"/>
```

#### Customized Constraints

If you want more sophisticated constraint, you could specify an object which implements the com.potix.zul.html.Constraint interface.

```
<zk>
<zscript><![CDATA[
Constraint ctt = new Constraint() {
   public void validate(Component comp, Object value) throws WrongValueException {
      if (((Integer)value).intValue() < 100)
         throw new WrongValueException(comp, "At least 100 must be specified");
   }
   public String getValidationScript() {return null;}
   boolean isClientComplete() {return false;}
}
]]></zscript>
<intbox constraint="${ctt}"/>
<window height="400px"/>
</zk>
```

You could implement your constraint into a Java class, say my.EmailValidator, then:

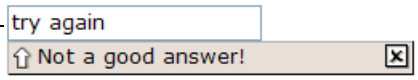
```
<?taglib uri="/WEB-INF/tld/web/core.dsp.tld" prefix="c"?>
```

```
<textbox constraint="${c:new('my.EmailValidator')}"/>
```

### com.potix.zk.ui.WrongValueException

In the above example, we use `com.potix.zk.ui.WrongValueException` to denote an error. As depicted, you have to specify the first argument with the component that causes the error, and then the second argument with the error message.

You could throw this exception anytime, such as when an `onChange` event is received as follows.




```
<textbox>
   <attribute name="onChange">
      if (!self.value.equals("good")) {
         self.value = "try again";
         throw new WrongValueException(self, "Not a good answer!");
      }
   </attribute>
</textbox>
```

### The `onChange` Event

An input control notifies the application with the `onChange` event if its content is changed by the user.

Notice that, when the `onChange`'s event listener is invoked, the value has been set. Thus, it is too late if you want to reject illegal value in the `onChange`'s event listener, unless you restore the value properly. Rather, it is recommended to use a constraint as described in the **Customized Constraints** section.

### The `onChanging` event

An input control also notifies the application with the `onChanging` event, when user is changing the content.

Notice that, when the `onChanging`'s listener is invoked, the value is not set yet. In other worlds, the value attribute still remain in the old value. To retrieve what the user is entering, you have to access the value attribute of the event as follows.

```
<grid>
   <rows>
      <row>The onChanging textbox:
         <textbox onChanging="copy.value = event.value"/></row>
      <row>Instant copy:
         <textbox id="copy" readonly="true"/></row>
   </rows>
</grid>
```

It is too early if you want to reject illegal value in the `onChanging`'s event listener,

because user may not complete the change yet. Rather, it is recommended to use a constraint as described in the **Customized Constraints** section.

### Calendar

A calendar displays a 'flat' calendar and allows user to select a day from it.

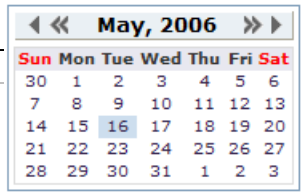

```
<hbox>
   <calendar id="cal" onChange="in.value = cal.value"/>
   <datebox id="in" onChange="cal.value = in.value"/>
</hbox>
```

#### The `value` Attribute and the `onChange` Event

Like input controls, calendar supports the value attribute to let developers set and retrieve the selected day. In additions, developers could listen to the `onChange` event to process it immediately, if necessary.

#### The `compact` Attribute

A calendar supports two different layouts and you can control it by use of the `compact` attribute.




```
<calendar compact="true"/>
```

The default value depends on the current Locale.

### Slider

A slider is used to let user specifying a value by scrolling.

```
<slider id="slider" onScroll="Audio.setVolume(slider.curpos)"/>
```

A slider accepts a range of value starting from 0 to 100. You could change the maximal allowed value by the `maxpos` attribute.

### Timer

A timer is an invisible component used to send the `onTimer` event to the server at the specified time or period. You could control a timer by the `start` and `stop` methods.

```
<window title="Timer demo" border="normal">
   <label id="now"/>
   <timer id="timer" delay="1000" repeats="true"
      onTimer="now.setValue(new Date().toString())"/>
   <separator bar="true"/>
```

Mon Dec 12 21:17:38 CST 2005

Stops timer | Starts timer

```
   <button label="Stops timer" onClick="timer.stop()"/>
   <button label="Starts timer" onClick="timer.start()"/>
</window>
```

## Windows

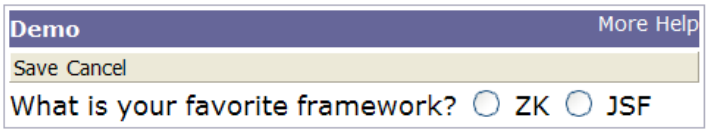
A window is, like HTML DIV tag, used to group components. Unlike other components, a window has the following characteristics.

- A window is an owner of an ID space. Any component contained in a window, including itself, could be found by use of the `getFellow` method, if it is assigned with an identifier.
- A window could be overlapped, popup, and embedded.
- A window could be a modal dialog.

### Titles and Captions

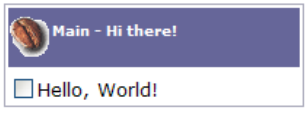
A window might have a title, a caption and a border. The title is specified by the `title` attribute. The caption is specified by declaring a child component called `caption`. All children of the `caption` component will appear to the right side of the title.

```
<window title="Demo" border="normal" width="350px">
   <caption>
      <toolbarbutton label="More"/>
      <toolbarbutton label="Help"/>
   </caption>
   <toolbar>
      <toolbarbutton label="Save"/>
      <toolbarbutton label="Cancel"/>
   </toolbar>
   What is your favorite framework?
   <radiogroup>
      <radio label="ZK"/>
      <radio label="JSF"/>
   </radiogroup>
</window>
```




You could also specify a label and an image to a caption, and then the appearance is as follows.

```
<window id="win" title="Main" border="normal" width="200px">
   <caption image="/img/coffee.gif" label="Hi there!"/>
   <checkbox label="Hello, World!"/>
</window>
```

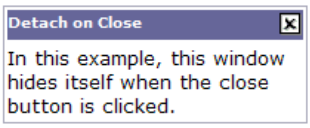



### The `closable` Attribute

By setting the `closable` attribute to true, you enable a close button for a window, such that

user could close the window by clicking the button. Once user clicks on the `close` button, the window becomes invisible, and an `onClose` event is sent to the window. It is processed by the `onClose` method of Window. Then, `onClose`, by default, detaches the window itself.

You can override it to do whatever you want. Or, you registered a listener to change the default behavior. For example, you might choose to hide rather than close.




```
<window closable="true" title="Detach on Close" border="normal" width="200px"
onClose="self.visible = false; event.stopPropagation();">
   In this example, this window hides itself when the close button is clicked.
</window>
```

Notice that `event.stopPropagation()` must be called to prevent Window.onClose() being called.

### Borders

The `border` attribute controls what CSS to apply to a window. In other words, a different value generates a different CSS class for the window. Moreover, whether a window is embedded, overlapped, popup and modal also affects what CSS class to use.

| The border | Embedded | Overlapped | Popup | Modal |
|---|---|---|---|---|
| normal | embedded | overlapped | popup | modal |
| none | embedded-none | overlapped-none | popup-none | modal-none |
| xyz | embedded-xyz | overlapped-xyz | popup-xyz | modal-xyz |

Currently, only the style classes for `normal` and `none` are defined. Depending on your preferences, you might add more style classes or customize the standard ones.

### Overlapped, Popup, Modal and Embedded

A window could be in one of four different modes: overlapped, popup, modal and embedded. By default, it is in the embedded mode. You could change the mode by use of the `doOverlapped`, `doPopup`, `doModal` and `doEmbedded` methods, depicted as follows.

```
<zk>
   <window id="win" title="Hi!" border="normal" width="200px">
      <caption>
         <toolbarbutton label="Close" onClick="win.setVisible(false)"/>
      </caption>
      <checkbox label="Hello, Wolrd!"/>
   </window>

   <button label="Overlap" onClick="win.doOverlapped();"/>
   <button label="Popup" onClick="win.doPopup();"/>
   <button label="Modal" onClick="win.doModal();"/>
   <button label="Embed" onClick="win.doEmbedded();"/>
```

```
</zk>
```

#### Embedded

An embedded window is placed inline with other components. In this mode, you cannot change its position, since the position is decided by the browser.

#### Overlapped

An overlapped window is overlapped with other components, such that users could drag it around and developer could set its position by the `setLeft` and `setTop` methods.

#### Popup

A popup window is similar to overlapped windows, except it is automatically closed when user clicks on any component other than the popup window itself or any of its descendants. As its name suggested, it is designed to implement popup windows.

#### Modal

A modal window (aka., a modal dialog) is similar to overlapped windows, except it suspends the event processing thread until one of the `endModal`, `doEmbedded`, `doOverlapped` and `doPopup` methods is called.

In addition to suspending the event processing thread, it disables components not belonging to the modal window.

A modal window is positioned automatically at the center of the browser, so you cannot control its position.

### Common Dialogs

The XUL component set supports the following common dialogs to simplify some common tasks.

#### The `com.potix.zul.html.Messagebox` Class

A set of utilities to show message boxes. It is typically used to alert user when an error occurs, or to prompt user for an decision.

```
if (Messagebox.show("Remove this file?", "Remove?", Messagebox.YES | Messagebox.NO,
Messagebox.QUESTION) == Messagebox.YES) {
   ...//remove the file
}
```

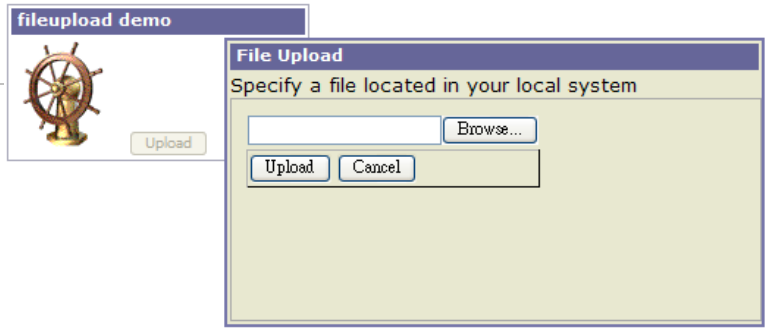
Since it is common to alert user for an error, a global function called `alert` is added for zscript. The `alert` function is a shortcut of the `show` method in the `Messagebox` class. In other words, The following two statements are equivalent.

```
<zscirpt>
   alert("Wrong");
   Messagebox.show("Wrong");
</zscript>
```

### The `com.potix.zul.html.Fileupload` Class

It contains utilities to prompt a user for uploading a file from the browser. The `get` method will show a dialog that prompts the user at the browser for specifying a file for uploading. It won't return until user has uploaded a file or presses the cancel button.

```
<window title="Fileupload Demo" border="normal">
   <image id="image"/>
   <button label="Upload">
      <attribute name="onClick">{
         Object media = Fileupload.get();
         if (media instanceof com.potix.image.Image)
            image.setContent(media);
         else if (media != null)
            Messagebox.show("Not an image: "+media, "Error",
               Messagebox.OK, Messagebox.ERROR);
      }</attribute>
   </button>
</window>
```




## The Box Model

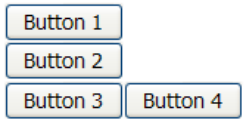
Components: `vbox`, `hbox` and `box`.

The box model of XUL is used to divide a portion of the display into a series of boxes. Components inside of a box will orient themselves horizontally or vertically. By combining a series of boxes and separators, you can control the layout of the visual representation.

A box can lay out its children in one of two orientations, either horizontally or vertically. A horizontal box lines up its components horizontally and a vertical box orients its components vertically. You can think of a box as one row or one column from an HTML table.

Some examples are shown as follows.

```
<zk>
   <vbox>
      <button label="Button 1"/>
      <button label="Button 2"/>
   </vbox>
   <hbox>
      <button label="Button 3"/>
      <button label="Button 4"/>
   </hbox>
```



```
</zk>
```

The `hbox` component is used to create a horizontally oriented box. Each component placed in the `hbox` will be placed horizontally in a row. The `vbox` component is used to create a vertically oriented box. Added components will be placed underneath each other in a column.

There is also a generic box component which defaults to horizontal orientation, meaning that it is equivalent to the `hbox`. However, you can use the `orient` attribute to control the orientation of the box. You can set this attribute to the value `horizontal` to create a horizontal box and `vertical` to create a vertical box.

Thus, the two lines below are equivalent:

```
<vbox>
<box orient="vertical">
```

You can add as many components as you want inside a box, including other boxes. In the case of a horizontal box, each additional component will be placed to the right of the previous one. The components will not wrap at all so the more components you add, the wider the window will be. Similarly, each element added to a vertical box will be placed underneath the previous one.

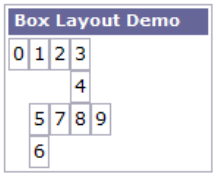
### The `spacing` Attribute

You could control the spacing among children of the `box` control. For example, the following example puts `5em` at both the upper margin and the lower margin. Notice: the total space between two input fields is `10em`.

```
<vbox spacing="5em">
   <textbox/>
   <datebox/>
</vbox>
```

Another example illustrated an interesting layout by use of zero spacing.

```
<window title="Box Layout Demo" border="normal">
   <hbox spacing="0">
      <window border="normal">0</window>
      <vbox spacing="0">
         <hbox spacing="0">
            <window border="normal">1</window>
            <window border="normal">2</window>
            <vbox spacing="0">
               <window border="normal">3</window>
               <window border="normal">4</window>
            </vbox>
         </hbox>
         <hbox spacing="0">
            <vbox spacing="0">
               <window border="normal">5</window>
               <window border="normal">6</window>
            </vbox>
```



```
            <window border="normal">7</window>
            <window border="normal">8</window>
            <window border="normal">9</window>
         </hbox>
      </vbox>
   </hbox>
</window>
```

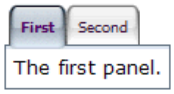
## Tab Boxes

Components: `tabbox`, `tabs`, `tab`, `tabpanels` and `tabpanel`.

A tab box allows developers to separate a large number of components into several groups, and show one group each time, such that the user interface won't be too complicate to read. There is only one group (aka., a panel) is visible at the same time. Once the tab of an invisible group is clicked, it becomes visible and the previous visible group becomes invisible.

The generic syntax of tab boxes is as follows.

```
<tabbox>
   <tabs>
      <tab label="First"/>
      <tab label="Second"/>
   </tabs>
   <tabpanels>
      <tabpanel>The first panel.</tabpanel>
      <tabpanel>The second panel</tabpanel>
   </tabpanels>
</tabbox>
```




- `tabbox`: The outer box that contains the tabs and tab panels.
- `tabs`: The container for the tabs, i.e., a collection of `tab` components.
- `tab`: A specific tab. Clicking on the tab brings the tab panel to the front. You could put a label and an image on it.
- `tabpanels`: The container for the tab panels, i.e., a collection of `tabpanel` components.
- `tabpanel`: The body of a single tab panel. You would place the content for a group of components within a tab panel. The first `tabpanel` corresponds to the first `tab`, the second `tabpanel` corresponds to the second `tab` and so on.

The currently selected tab component is given an additional `selected` attribute which is set to `true`. This is used to give the currently selected tab a different appearance so that it will look selected. Only one tab will have a `true` value for this attribute at a time.

There are two way to change the selected tab by Java codes. They are equivalent as shown below.

```
tab1.setSelected(true);
tabbox.setSelectedTab(tab1);
```

Of course, you can assign true to the selected attribute directly.

```
<tab label="My Tab" selected="true"/>
```

If none of tabs are selected, the first one is selected automatically.

### Nested Tab Boxes

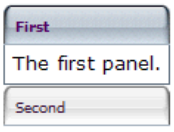
A tab panel could contain anything including another tab boxes.

```
<tabbox>
   <tabs>
      <tab label="First"/>
      <tab label="Second"/>
   </tabs>
   <tabpanels>
      <tabpanel>
      The first panel.
      <tabbox>
         <tabs>
            <tab label="Nested 1"/>
            <tab label="Nested 2"/>
            <tab label="Nested 3"/>
         </tabs>
         <tabpanels>
            <tabpanel>The first nested panel</tabpanel>
            <tabpanel>The second nested panel</tabpanel>
            <tabpanel>The third nested panel</tabpanel>
         </tabpanels>
      </tabbox>
      </tabpanel>
      <tabpanel>The second panel</tabpanel>
   </tabpanels>
</tabbox>
```


### The Accordion Tab Boxes

Tab boxes supports two molds: `default` and `accordion`. The effect of the `accordion` mold is as follows.

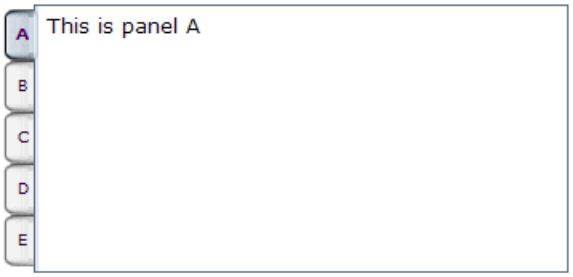
```
<tabbox mold="accordion">
   <tabs>
      <tab label="First"/>
      <tab label="Second"/>
   </tabs>
   <tabpanels>
      <tabpanel>The first panel.</tabpanel>
      <tabpanel>The second panel</tabpanel>
   </tabpanels>
</tabbox>
```



### The `orient` Attribute

Developers can control whether the tabs are located by use of the `orient` attribute. By default, it is `horizontal`. You can change it to `vertical`, and the effect is as follows.

```
<tabbox width="400px" orient="vertical">
   <tabs>
      <tab label="A"/>
      <tab label="B"/>
      <tab label="C"/>
      <tab label="D"/>
      <tab label="E"/>
   </tabs>
   <tabpanels>
      <tabpanel>This is panel A</tabpanel>
      <tabpanel>This is panel B</tabpanel>
      <tabpanel>This is panel C</tabpanel>
      <tabpanel>This is panel D</tabpanel>
      <tabpanel>This is panel E</tabpanel>
   </tabpanels>
</tabbox>
```




### Create-on-Select for Tab Panels

As illustrated below, you could listen to the `onSelect` event, and then fulfill the content of the panel when it is selected.

```
<tabbox id="tabbox" width="400" onSelect="load()" mold="accordion">
   <tabs>
      <tab label="Preload"/>
      <tab label="OnDemand"/>
   </tabs>
   <tabpanels>
      <tabpanel>
   This panel is pre-loaded.
      </tabpanel>
      <tabpanel>
      </tabpanel>
   </tabpanels>
   <zscript><![CDATA[
   void load() {
      Tabpanel panel = tabbox.getSelectedPanel();
      if (panel != null && panel.getChildren().isEmpty())
         new Label("Second panel is loaded").setParent(panel);
   }
   ]]></zscript>
</tabbox>
```

# More Layout Components

## Separators and Spaces

Components: `separator` and `space`.

A separator is used to insert a space between two components. There are several ways to customize the separator.

1. By use of the `orient` attribute, you could specify a vertical separator or a horizontal separator. By default, it is a horizontal separator, which inserts a line break. On the other hand, a vertical separator inserts a white space. In addition, `space` is a variant of `separator` whose default orientation is vertical.
2. By use of the `bar` attribute, you could control whether to show a horizontal or vertical line between component.
3. By use of the `spacing` attribute, you could control the size of spacing.

line 1 by separator
line 2 by separator
line 3 by separator | another piece
line 4 by separator | another piece

```
<window>
   line 1 by separator
   <separator/>
   line 2 by separator
   <separator/>
   line 3 by separator<space bar="true"/>another piece
   <separator spacing="20px"/>
   line 4 by separator<space bar="true" spacing="20px"/>another piece
</window>
```

## Group boxes

Components: `groupbox`.

A group box is used to group components together. A border is typically drawn around the components to show that they are related.

The label across the top of the group box can be created by using the `caption` component. It works much like the HTML legend element.

Unlike windows, a group box is not an owner of the ID space. It cannot be overlapped or popup.

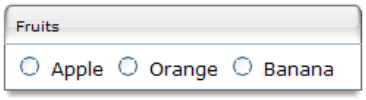
Fruits
Apple Orange Banana

```
<groupbox width="250px">
   <caption label="Fruits"/>
   <radiogroup>
      <radio label="Apple"/>
      <radio label="Orange"/>
      <radio label="Banana"/>
   </radiogroup>
```

```
</groupbox>
```

In addition to the `default` mold, the group box also supports the `3d` mold. If the 3d mold is used, it works similar to a simple-tab tab box. First, you could control whether its content is visible by the `open` attribute. Similarly, you could create the content of a group box when the `onOpen` event is received.

```
<groupbox mold="3d" open="true" width="250px">
   <caption label="fruits"/>
   <radiogroup>
      <radio label="Apple"/>
      <radio label="Orange"/>
      <radio label="Banana"/>
   </radiogroup>
</groupbox>
```




### Grids

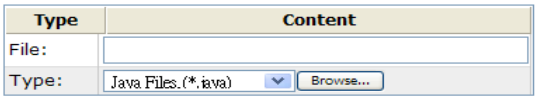
Components: `grid`, `columns`, `column`, `rows` and `row`.

A grid contains components that are aligned in rows like tables. Inside a grid, you declare two things, the columns, that define the header and column attributes, and the rows, that provide the content.

To declare a set of rows, use the `rows` component, which should be a child element of `grid`. Inside that you should add `row` components, which are used for each row. Inside the row element, you should place the content that you want inside that row. Each child is a column of the specific row.

Similarly, the columns are declared with the `columns` component, which should be placed as a child element of the grid. Unlike `row` is used to hold the content of each row, `column` declares the common attributes of each column, such as the width and alignment, and and optional headers, i.e., label and/or image.

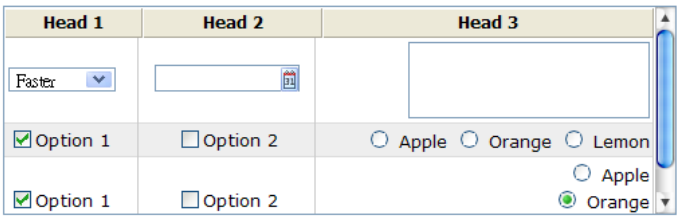
```
<grid>
   <columns>
      <column label="Type"/>
      <column label="Content"/>
   </columns>
   <rows>
      <row>
         <label value="File:"/>
         <textbox width="99%"/>
      </row>
      <row>
         <label value="Type:"/>
         <hbox>
            <listbox rows="1" mold="select">
               <listitem label="Java Files,(*.java)"/>
               <listitem label="All Files,(*.*)"/>
```



```
            </listbox>
            <button label="Browse..."/>
         </hbox>
      </row>
   </rows>
</grid>
```

### Scrollable Grid

A grid could be scrollable if you specify the height attribute and there is not enough space to show all data.




```
<grid id="grid" width="500px" height="130px">
   <columns>
      <column label="Head 1"/>
      <column label="Head 2" align="center"/>
      <column label="Head 3" align="right"/>
   </columns>
   <rows>
      <row>
         <listbox mold="select">
            <listitem label="Faster"/>
            <listitem label="Fast"/>
            <listitem label="Average"/>
         </listbox>
         <datebox/>
         <textbox rows="2"/>
      </row>
      <row>
         <checkbox checked="true" label="Option 1"/>
         <checkbox label="Option 2"/>
         <radiogroup>
            <radio label="Apple"/>
            <radio label="Orange" checked="true"/>
            <radio label="Lemon"/>
         </radiogroup>
      </row>
      <row>
         <checkbox checked="true" label="Option 1"/>
         <checkbox label="Option 2"/>
         <radiogroup orient="vertical">
            <radio label="Apple"/>
            <radio label="Orange" checked="true"/>
            <radio label="Lemon"/>
         </radiogroup>
      </row>
   </rows>
</grid>
```

### Sorting

Grids support the sorting of rows directly. To enable the ascending order for a particular column, you assign a `java.util.Comparator` instance to the `sortAscending` attribute of the column. Similarly, you assign a comparator to the `sortDescending` attribute to enable the descending order.

As illustrated below, you first implement a comparator that compares any two rows of the grid, and then assign its instances to the `sortAscending` and `sortDescending` attributes. Notice: the `compareTo` method is called with two `com.potix.zul.html.Row` instance.

```
<zk>
   <zscript>
      class MyRowComparator implements Comparator {
         public MyRowComparator(boolean asending) {
         ...
         }
         public int compareTo(Object o1, Object o2) {
            Row r1 = (Row)o1, r2 = (Row)o2;
            ....
         }
      }
      Comparator asc = new MyRowComparator(true);
      Comparator dsc = new MyRowComparator(false);
   </zscript>
   <grid>
      <columns>
         <column sortAscending="${asc}" sortDescending="${dsc}"/>
...
```

### The `sortDirection` Attribute

The `sortDirection` attribute controls whether to show an icon at the client to indicate the order of a particular column. If rows are sorted before adding to the grid, you shall set this attribute explicitly.

```
<column sortDirection="ascending"/>
```

Then, it is maintained automatically by grids as long as you assign the comparators to the corresponding column.

### The `onSort` Event

When you assign at least one comparator to a column, an `onSort` event is sent to the server if user clicks on it. The `column` component implements a listener to automatically sort rows based on the assigned comparator.

If you prefer to handle it manually, you can add your own listener to the column for the `onSort` event. Alternatively, you can override the `sort` method, see below.

### The `sort` Method

The `sort` method is the underlying implementation of the default `onSort` event listener. It is also useful if you wan to sort the rows by Java codes. For example, you might have to call this method after adding rows (assuming not in the proper order).

```
Row row = new Row();
row.setParent(listbox);
row.appendChild(...);
...
if (!"natural".column.getSortDirection())
   column.sort("ascending".equals(column.getSortDirection()));
```

The default sorting algorithm is quick-sort (by use of the `sort` method from the `com.potix.zk.ui.Components` class). You might override it with your own implementation.

### Toolbars

Components: `toolbar` and `toolbarbutton`.

A toolbar is used to place a series of buttons, such as toolbar buttons. The toolbar buttons could be used without toolbars, so a toolbar could be used without tool buttons. However, tool buttons change their appearance if they are placed inside a toolbar.

The toolbar has two orientation: `horizontal` and `vertical`. It controls how the buttons are placed.

```
<toolbar>
   <toolbarbutton label="button1"/>
   <toolbarbutton label="button2"/>
</toolbar>
```

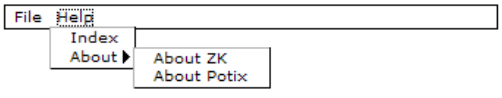

## Menu bars

Components: `menubar`, `menupopup`, `menu`, `menuitem` and `menuseparator`.

A menu bar contains a collection of menu items and sub menus. A sub menu contains a collection of menu items and other sub menus. They, therefore, constructs a tree of menu items that user could select to execute.

An example of menu bars is as follows.

```
<menubar>
   <menu label="File">
      <menupopup>
         <menuitem label="New"/>
         <menuitem label="Open"/>
         <menuseparator/>
```



```
         <menuitem label="Exit"/>
      </menupopup>
   </menu>
   <menu label="Help">
      <menupopup>
         <menuitem label="Index"/>
         <menu label="About">
            <menupopup>
               <menuitem label="About ZK"/>
               <menuitem label="About Potix"/>
            </menupopup>
         </menu>
      </menupopup>
   </menu>
</menubar>
```

- `menubar`: The topmost container for a collection of menu items (`menuitem`) and menus (`menu`).
- `menu`: The container of a popup menu. It also defines the label to be displayed at part of its parent. When user clicks on the label, the popup menu appears.
- `menupopup`: A container for a collection of menu items (`menuitem`) and menus (`menu`). It is a child of `menu` and appears when the label of `menu` is clicked.
- `menuitem`: An individual command on a menu. This could be placed in a menu bar, or a popup menu.
- `menuseparator`: A separator bar on a menu. This would be placed in a popup menu.

**Execute a Menu Command**

A menu command is associated with a menu item. There are two ways to associate a command to it: the `onClick` event and the `href` attribute. If a event listener is added for a menu item for the `onClick` event, the listener is invoked when the item is clicked.

```
<menuitem onClick="draft.save()"/>
```

On the other hand, you could specify the `href` attribute to hyperlink to the specified URL when a menu item is clicked.

```
<menuitem href="/edit"/>
<menuitem href="http://zk1.sourceforge.net"/>
```

If both of the event listener and `href` are specified, they will be executed. However, when the event listener get executed in the server, the browser might already change the current URL to the specified one. Thus, all responses generated by the event listener will be ignored.

### Use Menu Items as Check Boxes

A menu item could be used as a check box. The `checked` attribute denotes whether this menu item is checked. If checked, a check icon is appeared in front of the menu item.

In addition to programming the `checked` attribute, you could specify the `autoCheck` attribute to be `true`, such that the checked attribute is toggled automatically when user clicks the menu item.

```
<menuitem label="" autoCheck="true"/>
```

### The `autoPopup` Attribute

By default, the popup menu is opened when user clicks on it. You might change this to automatically popup menu when the mouse moves over it. This is done by setting the `autoPopup` attribute to true.

```
<menubar autoPopup="true">
   ...
</menubar>
```

### More Menu Features

Like `box`, you could control the orientation of a menu by use of the `orient` attribute. By default, the orientation is `horizontal`.

Like other components, you could change the menu dynamically, including attributes and creating sub menus. Refer to `menu.zul` under the `test` directory in `zkdemo`.

## List Boxes

Components: `listbox`, `listitem`, `listcell`, `listhead` and `listheader`.

A list box is used to display a number of items in a list. The user may select an item from the list.

The simplest format is as follows. It is a single-column and single-selection list box.

```
<listbox>
   <listitem label="Butter Pecan"/>
   <listitem label="Chocolate Chip"/>
   <listitem label="Raspberry Ripple"/>
</listbox>
```

Butter Pecan
Chocolate Chip
Raspberry Ripple

Listbox has two molds: `default` and `select`. If the `select` mold is used, the HTML's `SELECT` tag is generated instead.

```
<listbox mold="select">...</listbox>
```

Butter Pecan
Chocolate Chip
Raspberry Ripple

Notice: if `mold` is "select", `rows` is "1", and none of items is marked as selected, the

browser assumes the first item is selected. In other words, it won't send the `onSelect` event, if user clicks the first item in this case.

In addition to label, you can assign an application-specific value to each item using the `setValue` method.

**Mouseless Entry** **`listbox`**

- `UP` and `DOWN` to move the selection up and down one list item.
- `PgUp` and `PgDn` to move the selection up and down in a step of one page.
- `HOME` to move the selection to the first item, and `END` to the last item.
- `Ctrl+UP` and `Ctrl+DOWN` to move the focus up and down one list item without changing the selection.
- `SPACE` to select the item of the focus.

### Multi-Column List Boxes

The list box also supports multiple columns. When user selects an item, the entire row is selected.

To specify a multi-column list, you need to specify the listcell components as collumns of each listitem (as a row).

```
<listbox width="200px">
   <listitem>
      <listcell label="George"/>
      <listcell label="House Painter"/>
   </listitem>
   <listitem>
      <listcell label="Mary Ellen"/>
      <listcell label="Candle Maker"/>
   </listitem>
   <listitem>
      <listcell label="Roger"/>
      <listcell label="Swashbuckler"/>
   </listitem>
</listbox>
```

| | |
|---|---|
| George | House Painter |
| Mary Ellen | Candle Maker |
| Roger | Swashbuckler |

### Column Headers

You could specify the column headers by use of `listhead` and `listheader` as follows[27]. In addition to label, you could specify an image as the header by use of the `image` attribute.

```
<listbox width="200px">
   <listhead>
      <listheader label="Name"/>
      <listheader label="Occupation"/>
   </listhead>
```

| Name | Occupation |
|---|---|
| George | House Painter |
| Mary Ellen | Candle Maker |
| Roger | Swashbuckler |

27 This feature is a bit different from XUL, where listhead and listheader are used.

```
...
</listbox>
```

### Column Footers

You could specify the column footers by use of `listfoot` and `listfooter` as follows. Notice that the order of `listhead` and `listfoot` doesn't matter. Each time a `listhead` instance is added to a list box, it must be the first child, and a `listfoot` instance the last child.

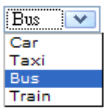
```
<listbox width="200px">
   <listhead>
      <listheader label="Population"/>
      <listheader align="right" label="%"/>
   </listhead>
   <listitem id="a" value="A">
      <listcell label="A. Graduate"/>
      <listcell label="20%"/>
   </listitem>
   <listitem id="b" value="B">
      <listcell label="B. College"/>
      <listcell label="23%"/>
   </listitem>
   <listitem id="c" value="C">
      <listcell label="C. High School"/>
      <listcell label="40%"/>
   </listitem>
   <listitem id="d" value="D">
      <listcell label="D. Others"/>
      <listcell label="17%"/>
   </listitem>
   <listfoot>
      <listfooter label="More or less"/>
      <listfooter label="100%"/>
   </listfoot>
</listbox>
```

| Population | % |
|---|---|
| A. Graduate | 20% |
| B. College | 23% |
| C. High School | 40% |
| D. Others | 17% |
| More or less | 100% |

### Drop-Down List

You could create a drop-down list by specifying the select mold and single row. Notice you cannot use multi-column for the drop-down list.

```
<listbox mold="select" rows="1">
   <listitem label="Car"/>
   <listitem label="Taxi"/>
   <listitem label="Bus" selected="true"/>
   <listitem label="Train"/>
</listbox>
```



### Multiple Selection

When user clicks on a list item, the whole item is selected and the `onSelect` event is sent back to the server to notify the application. You could control whether a list box allows multiple selections by setting the `multiple` attribute to true. The default value is false.

### Sorting

List boxes support sorting of list items directly. There are a few ways to enable the sorting of a particular column. The simplest way is to set the `sort` attribute of the list header to `auto` as follows. Then, the column that the list header is associated with is sortable based on the label of each list cell of the specified column.

```
<zk>
   <listbox width="200px">
      <listhead>
         <listheader label="name" sort="auto"/>
         <listheader label="gender" sort="auto"/>
      </listhead>
      <listitem>
         <listcell label="Mary"/>
         <listcell label="FEMALE"/>
      </listitem>
      <listitem>
         <listcell label="John"/>
         <listcell label="MALE"/>
      </listitem>
      <listitem>
         <listcell label="Jane"/>
         <listcell label="FEMALE"/>
      </listitem>
      <listitem>
         <listcell label="Henry"/>
         <listcell label="MALE"/>
      </listitem>
   </listbox>
</zk>
```

| name | gender |
|---|---|
| Henry | MALE |
| Jane | FEMALE |
| John | MALE |
| Mary | FEMALE |

#### The `sortAscending` and `sortDescending` Attributes

If you prefer to sort list items in different ways, you can assign a `java.util.Comparator` instance to the `sortAscending` and/or `sortDescending` property. Once assigned, the list items can be sorted in the ascending and/or descending order with the comparator you assigned.

The invocation of the `sort` attribute with `auto` actually assign two comparators to the `sortAsceding` and `sortDescending` automatically. You can override any of them by assigning another comparator to it.

For example, assume you want to sort based on the value of list items, rather than list cell's label, then you assign an instance of `ListitemComparator` to these attributes as follows.

```
<zscript>
   Comparator asc = new ListitemComarator(-1, true, true);
   Comparator dsc = new ListitemComarator(-1, false, true);
</zscript>
<listbox>
   <listhead>
      <listheader sortAscending="${asc}" sortDescending="${dsc}"/>
...
```

#### The `sortDirection` Attribute

The `sortDirection` attribute controls whether to show an icon at the client to indicate the order of the particular column. If list items are sorted before adding to the list box, you shall set this attribute explicitly.

```
<listheader sortDirection="ascending"/>
```

Then, it is maintained automatically by list boxes as long as you assign the comparator to the corresponding list header.

#### The `onSort` Event

When you assign at least one comparator to a list header, an `onSort` event is sent to the server if user clicks on it. The list header implements a listener to handle the sorting automatically.

If you prefer to handle it manually, you can add your listener to the list header for the `onSort` event. Alternatively, you can override the `sort` method, see below.

#### The `sort` Method

The `sort` method is the underlying implementation of the default `onSort` event listener. It is also useful if you wan to sort the list items by Java codes. For example, you might have to call this method after adding items (assuming not in the proper order).

```
new Listem("New Stuff").setParent(listbox);
if (!"natural".header.getSortDirection())
   header.sort("ascending".equals(header.getSortDirection()));
```

The default sorting algorithm is quick-sort (by use of the `sort` method from the `com.potix.zk.ui.Components` class). You might override it with your own implementation.

## More Attributes

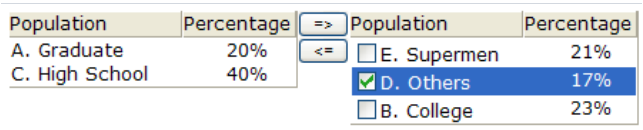
### The `rows` Attribute

The `rows` attribute is used to control how many rows are visible. By setting it to zero, the list box will resize itself to hold as many as items if possible.

### The `checkmark` Attribute

The `checkmark` attribute controls whether to display a checkbox or a radio button in front of each list item.

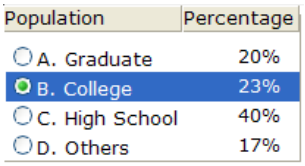
In the following example, you will see how a checkbox is added automatically, when you move a list item from the left list box to the right one. The checkbox is removed when you move a list item from right to left.




```
<hbox>
   <listbox id="src" rows="0" multiple="true" width="200px">
      <listhead>
         <listheader label="Population"/>
         <listheader label="Percentage"/>
      </listhead>
      <listitem id="a" value="A">
         <listcell label="A. Graduate"/>
         <listcell label="20%"/>
      </listitem>
      <listitem id="b" value="B">
         <listcell label="B. College"/>
         <listcell label="23%"/>
      </listitem>
      <listitem id="c" value="C">
         <listcell label="C. High School"/>
         <listcell label="40%"/>
      </listitem>
      <listitem id="d" value="D">
         <listcell label="D. Others"/>
         <listcell label="17%"/>
      </listitem>
   </listbox>
   <vbox>
      <button label="=&gt;" onClick="move(src, dst)"/>
      <button label="&lt;=" onClick="move(dst, src)"/>
   </vbox>
   <listbox id="dst" checkmark="true" rows="0" multiple="true" width="200px">
      <listhead>
         <listheader label="Population"/>
         <listheader label="Percentage"/>
      </listhead>
```

```
      <listitem id="e" value="E">
         <listcell label="E. Supermen"/>
         <listcell label="21%"/>
      </listitem>
   </listbox>
   <zscript>
void move(Listbox src, Listbox dst) {
   Listitem s = src.getSelectedItem();
   if (s == null)
      Messagebox.show("Select an item first");
   else
      s.setParent(dst);
}
   </zscript>
</hbox>
```

Notice that if the `multiple` attribute is false, the radio buttons are displayed instead, as depicted at the right.




### The maxlength Attribute

The `maxlength` attribute defines the maximal allowed characters being visible at the browser. By setting this attribute, you could make a narrower list box.

## Live Data

Like Swing[28], list boxes supports the live data. With live data, developers could separate the data from the view. In other words, developers needs only to provide the data by implementing `com.potix.zul.html.ListModel` interface. Rather than manipulating the list box directly. The benefits are two folds.

- It is easier to use different views to show the same set of data.
- The list box sends the data to the client only if it is visible. It saves a lot of network traffic if the amount of data is huge.

There are three steps to use the live data.

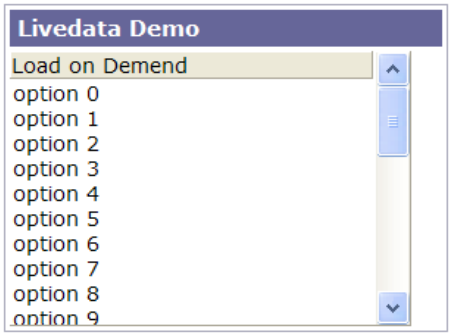
1. Prepare the data in the form of `ListModel`. ZK has a concrete implementation called `com.potix.zul.html.SimpleListModel`. for representing an array of objects.
2. Implement the `com.potix.zul.html.ListitemRenderer` interface for rendering an item of data into a row of the list box.
   - This is optional. If not specified, the default renderer is used to render the data into the first column.
   - You could implement different rederers for represent the same data in different views.

28 The concept is similar to Swing (`javax.swing.ListModel`).

3. Specify the data in the `model` attribute, and, optionally, the renderer in the `itemRenderer` attribute.

In the following example, we prepared a list model called `strset` for a list box, assigned it to a list box through the `model` attribute. Then, the list box will do the rest.

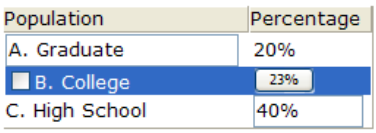
```
<window title="livedata demo" border="normal">
   <zscript>
      String[] data = new String[30];
      for(int j=0; j &lt; data.length; ++j) {
         data[j] = "option "+j;
      }
      ListModel strset = new SimpleListModel(data);
   </zscript>
   <listbox id="list" width="200px" rows="10"
model="${strset}">
      <listhead>
         <listheader label="Load on demend"/>
      </listhead>
   </listbox>
</window>
```




### List Boxes Contain Buttons

In theory, a list cell could contain any other components, as depicted below.

```
<listbox width="250px">
   <listhead>
      <listheader label="Population"/>
      <listheader label="Percentage"/>
   </listhead>
   <listitem value="A">
      <listcell><textbox value="A. Graduate"/></listcell>
      <listcell label="20%"/>
   </listitem>
   <listitem value="B">
      <listcell><checkbox label="B. College"/></listcell>
      <listcell><button label="23%"/></listcell>
   </listitem>
   <listitem value="C">
      <listcell label="C. High School"/>
      <listcell><textbox cols="8" value="40%"/></listcell>
   </listitem>
</listbox>
```




Notes:

1. Don't use a list box, when a grid is a better choice. The appearances of list boxes and grids are similar, but the list box shall be used only to represent a list of selectable items.
2. Users are usually confused if a list box contains editable components, such as

`textbox` and `checkbox`. A common question is what the text, that a user entered in a unselected item, means.

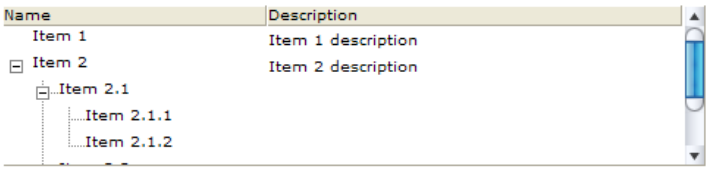
3. Due to the limitation of the browsers, users cannot select a piece of characters from the text boxes.

## Tree Controls

Components: `tree`, `treechildren`, `treeitem`, `treerow`, `treecell`, `treecols` and `treecol`.

A tree consists of two parts, the set of columns, and the tree body. The set of columns is defined by a number of `treecol` components, one for each column. Each column will appear as a header at the top of the tree. The second part, the tree body, contains the data to appear in the tree and is created with a `treechildren` component.

An example of a tree control is as follows.




```
<tree id="tree" rows="5">
   <treecols>
      <treecol label="Name"/>
      <treecol label="Description"/>
   </treecols>
   <treechildren>
      <treeitem>
         <treerow>
            <treecell label="Item 1"/>
            <treecell label="Item 1 description"/>
         </treerow>
      </treeitem>
      <treeitem>
         <treerow>
            <treecell label="Item 2"/>
            <treecell label="Item 2 description"/>
         </treerow>
         <treechildren>
            <treeitem>
               <treerow>
                  <treecell label="Item 2.1"/>
               </treerow>
               <treechildren>
                  <treeitem>
                   <treerow>
                          <treecell label="Item 2.1.1"/>
                   </treerow>
                  </treeitem>
                  <treeitem>
                   <treerow>
                          <treecell label="Item 2.1.2"/>
                   </treerow>
                  </treeitem>
```

```
            </treechildren>
         </treeitem>
         <treeitem>
            <treerow>
               <treecell label="Item 2.2"/>
               <treecell label="Item 2.2 is something who cares"/>
            </treerow>
         </treeitem>
      </treechildren>
   </treeitem>
   <treeitem label="Item 3"/>
 </treechildren>
</tree>
```

- `tree`: This is the outer component of a tree control.
- `treecols`: This component is a placeholder for a collection of `treecol` components.
- `treecol`: This is used to declare a column of the tree. By using this comlumn, you can specify additional information such as the column header.
- `treechildren`: This contains the main body of the tree, which contain a collection of `treeitem` components.
- `treeitem`: This component contains a row of data (`treerow`), and an optional `treechildren`.
  - If the component doesn't contain a `treechildren`, it is a leaf node that doesn't accept any child items.
  - If it contains a `treechildren`, it is a branch node that might contain other items.
  - For a branch node, an +/- button will appear at the beginning of the row, such that user could open and close the item by clicking on the +/- button.
- `treerow`: A single row in the tree, which should be placed inside a `treeitem` component.
- `treecol`: A single cell in a tree row. This element would go inside a `treerow` component.

**Mouseless Entry** **tree**

- `UP` and `DOWN` to move the selection up and down one tree item.
- `PgUp` and `PgDn` to move the selection up and down in a step of one page.
- `HOME` to move the selection to the first item, and `END` to the last item.
- `RIGHT` to open a tree item, and `LEFT` to close a tree item.
- `Ctrl+UP` and `Ctrl+DOWN` to move the focus up and down one tree item without changing the selection.
- `SPACE` to select the item of the focus.

### The `open` Attribute and the `onOpen` Event

Each tree item has the `open` attribute used to control whether to display its child items. The default value is true. By setting this attribute to false, you could control what part of the tree is invisible.

```
<treeitem open="false">
```

When a user clicks on the +/- button, he opens the tree item and makes its children visible. The `onOpen` event is then sent to the server to notify the application.

### Multiple Selection

When user clicks on a tree item, the whole item is selected and the `onSelect` event is sent back to the server to notify the application. You could control whether a tree control allows multiple selections by setting the `multiple` attribute to true. The default value is false.

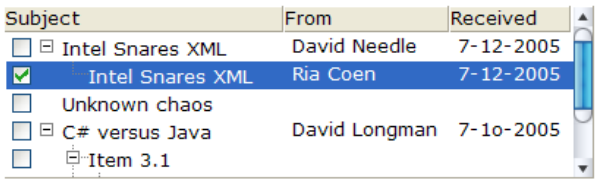
### Special Attributes

#### The `rows` Attribute

The `rows` attribute is used to control how many rows are visible. By setting it to zero, the tree control will resize itself to hold as many as items if possible.

#### The `checkmark` Attribute

The `checkmark` attribute controls whether to display a checkbox or a radio button in front of each tree item.




#### The `maxlength` Attribute

The `maxlength` attribute defines the maximal allowed characters being visible at the browser. By setting this attribute, you could make a narrower tree control.

### Create-on-Open for Tree Controls

As illustrated below, you could listen to the `onOpen` event, and then load the children of an tree item. Similarly, you could do create-on-open for group boxes.

```
<tree width="200px">
   <treecols>
      <treecol label="Subject"/>
```

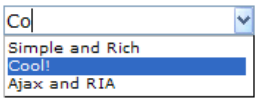
```
      <treecol label="From"/>
   </treecols>
   <treechildren>
      <treeitem open="false" onOpen="load()">
         <treerow>
            <treecell label="Intel Snares XML"/>
            <treecell label="David Needle"/>
         </treerow>
         <treechildren/>
      </treeitem>
   </treechildren>
   <zscript>
   void load() {
      Treechildren tc = self.getTreechildren();
      if (tc.getChildren().isEmpty()) {
         Treeitem ti = new Treeitem();
         ti.setLabel("New added");
         ti.setParent(tc);
      }
   }
   </zscript>
</tree>
```

## Comboboxes

Components: `combobox` and `comboitem`.

A combobox is a special text box that embeds a drop-down list. With comboboxes, users are allowed to select from a drop-down list, in addition to entering the text manually.

```
<combobox>
   <comboitem label="Simple and Rich"/>
   <comboitem label="Cool!"/>
   <comboitem label="Ajax and RIA"/>
</combobox>
```




**Mouseless Entry** **combobox**

- `Alt+DOWN` to pop up the list.
- `Alt+UP` or `ESC` to close the list.
- `UP` and `DOWN` to change the selection of the items from the list.

### The `autodrop` Attribute

By default, the drop-down list won't be opened until user clicks the button, or press `Alt+DOWN`. However, you could set the `autodrop` attribute to true, such that the drop-down list is opened as soon as user types any character. This is helpful for novice users, but it might be annoying for experienced users.

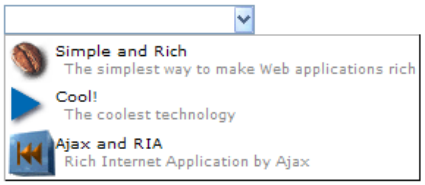
```
<combobox autodrop="true"/>
```

### The `description` Attribute

You could add a description to each combo item to make it more descriptive. In addition, you could assign an image to each combo item.

```
<combobox>
   <comboitem label="Simple and Rich" image="/img/coffee.gif"
   description="The simplest way to make Web applications rich"/>
   <comboitem label="Cool!" image="/img/corner.gif"
   description="The coolest technology"/>
   <comboitem label="Ajax and RIA" image="/img/cubfirs.gif"
   description="Rich Internet Application by Ajax"/>
</combobox>
```

Like other components that support images, you could use the `setImageContent` method to assign the content of a dynamically generated image to the `comboitem` component. Refer to the **Image** section for details.




### The `onOpen` Event

The `onOpen` event is sent to the application if user opens the drop-down list. By listening to the `onOpen` event, you could prepare the drop-down list only when it is needed.

```
<combobox id="combo" onOpen="prepare()"/>
<zscript>
   void prepare() {
      if (combo.getItemCount() == 0) {
         combo.appendItem("Simple and Rich");
         combo.appendItem("Cool!");
         combo.appendItem("Ajax and RIA");
      }
   }
</zscript>
```

The `appendItem` method is equivalent to create a combo item and then assign its parent to the como box.

### The `onChanging` Event

Since a combobox is also a text box, the `onChanging` event will be sent if you add a listener for it. By listening to this event, you could manipulate the drop-down list as the Google Suggests[29] does.

As illustrated below, you could fill the drop-down list based on what user is entering.

29 http://www.google.com/webhp?complete=1&hl=en

```
<combobox id="combo" autodrop="true" onChanging="suggest()"/>
<zscript>
   void suggest() {
      combo.clear();
      if (event.value.startsWith("A")) {
         combo.appendItem("Ace");
         combo.appendItem("Ajax");
         combo.appendItem("Apple");
      } else if (event.value.startsWith("B")) {
         combo.appendItem("Best");
         combo.appendItem("Blog");
      }
   }
</zscript>
```

Notice that, when the `onChanging` event is received, the content of the combobox is not changed yet. Thus, you cannot use the `value` attribute of the combobox. Rather, you shall use the `value` attribute of the event (`com.potix.zk.ui.event.InputEvent`).

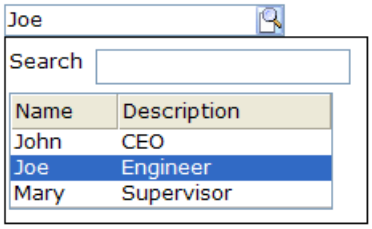
## Bandboxes

Components: `bandbox` and `bandpopup`.

A bandbox is a special text box that embeds a customizable popup window. Like comboboxes, a bandbox consists of an input box and a popup window. The popup window is opened automatically, when users presses `Alt+DOWN` or clicks the  button.

Unlike comboboxes, the popup window of a bandbox could be anything. It is designed to give developers the maximal flexibility. A typical use is to represent the popup window as a search dialog.

```
   <bandbox id="bd">
      <bandpopup>
<vbox>
      <hbox>Search <textbox/></hbox>
      <listbox width="200px"
      onSelect="bd.value=self.selectedItem.label; bd.closePopup();">
         <listhead>
            <listheader label="Name"/>
            <listheader label="Description"/>
         </listhead>
         <listitem>
            <listcell label="John"/>
            <listcell label="CEO"/>
         </listitem>
         <listitem>
            <listcell label="Joe"/>
            <listcell label="Engineer"/>
         </listitem>
         <listitem>
```



```
            <listcell label="Mary"/>
            <listcell label="Supervisor"/>
         </listitem>
      </listbox>
</vbox>
      </bandpopup>
   </bandbox>
```

**Mouseless Entry** **bandbox**

- `Alt+DOWN` to pop up the list.
- `Alt+UP` or `ESC` to close the list.
- `UP` and `DOWN` to change the selection of the items from the list.

### The `closePopup` Method

A popup window could contain any kind of components, so it is developer's job to copy the value from and close the popup if one of item is selected.

In the above example, we copy the selected item's label to the bandbox, and then close the popup by the following statement.

```
<listbox width="200px"
  onSelect="bd.value=self.selectedItem.label; bd.closePopup();">
```

### The `autodrop` Attribute

By default, the popup window won't be opened until user clicks the  button, or press `Alt+DOWN`. However, you could set the `autodrop` attribute to true, such that the popup is opened as soon as user types any character. This is helpful for novice users, but it might be annoying for experienced users.

```
<bandbox autodrop="true"/>
```

### The `onOpen` Event

The `onOpen` event is sent to the application if user opens the popup window. By listening to the `onOpen` event, you could prepare the popup window only when it is needed.

```
<bandbox id="band" onOpen="prepare()"/>
<zscript>
   void prepare() {
      if (band.getPopup() == null) {
         ...//create child elements
      }
   }
</zscript>
```

### The `onChanging` Event

Since a bandbox is also a text box, the `onChanging` event will be sent if you add a listener for it. By listening to this event, you could manipulate the popup window any way you like.

As illustrated below, you could fill the drop-down list based on what user is entering.

```
<bandbox id="band" autodrop="true" onChanging="suggest()"/>
<zscript>
   void suggest() {
      if (event.value.startsWith("A")) {
         ...//do something
      } else if (event.value.startsWith("B")) {
         ...//do another
      }
   }
</zscript>
```

Notice that, when the `onChanging` event is received, the content of the bandbox is not changed yet. Thus, you cannot use the `value` attribute of the bandbox. Rather, you shall use the `value` attribute of the event (`com.potix.zk.ui.event.InputEvent`).

## Drag and Drop

ZK allows a user to drag particular components around within the user interface. For example, dragging files to other directories, or dragging an item to the shopping cart to purchase.

A component is draggable if it can be dragged around. A component is droppable, if a user could drop a draggable component to it.

> **Note:** ZK does not assume any behavior about what shall take place after dropping. It is up to application developers by writing the `onDrop` event listener.
>
> If an application doesn't nothing, the dragged component is simply moved back where it is originated from.

### The `draggable` and `droppable` Attributes

With ZK, you could make a component draggable by assigning any value, other than `"false"`, to the draggable attribute. To disable it, assign it with `"false"`.

```
<image draggable="true"/>
```

Similarly, you could make a component droppable by assigning `"true"` to the droppable attribute.

```
<hbox droppable="true"/>
```

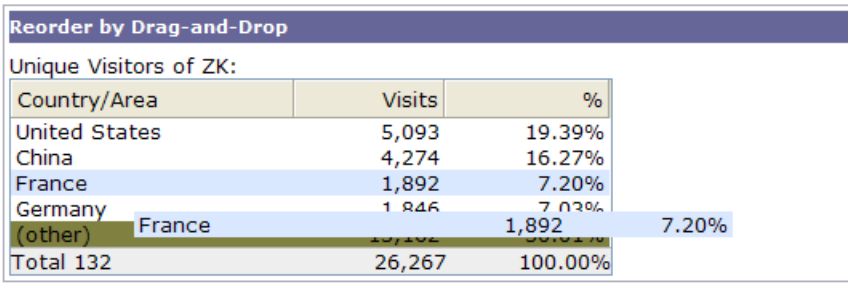
Then, user could drag a draggable component, and then drop it to a droppable component.

### The onDrop Event

Once user has dragged a component and dropped it to another component, the component that the user dropped the component to will be notified by the `onDrop` event. The `onDrop` event is an instance of `com.potix.ui.event.DropEvent`. By calling the `getDragged` method, you could retrieve what has been dragged (and dropped).

Notice that the target of the `onDrop` event is the droppable component, not the component being dragged.

The following is a simple example that allows users to reorder list items by drag-and-drop.




```
<window title="Reorder by Drag-and-Drop" border="normal">
   Unique Visitors of ZK:
   <listbox id="src" multiple="true" width="300px">
      <listhead>
         <listheader label="Country/Area"/>
         <listheader align="right" label="Visits"/>
         <listheader align="right" label="%"/>
      </listhead>
      <listitem draggable="true" droppable="true" onDrop="move(event.dragged)">
         <listcell label="United States"/>
         <listcell label="5,093"/>
         <listcell label="19.39%"/>
      </listitem>
      <listitem draggable="true" droppable="true" onDrop="move(event.dragged)">
         <listcell label="China"/>
         <listcell label="4,274"/>
         <listcell label="16.27%"/>
      </listitem>
      <listitem draggable="true" droppable="true" onDrop="move(event.dragged)">
         <listcell label="France"/>
         <listcell label="1,892"/>
         <listcell label="7.20%"/>
      </listitem>
      <listitem draggable="true" droppable="true" onDrop="move(event.dragged)">
         <listcell label="Germany"/>
         <listcell label="1,846"/>
         <listcell label="7.03%"/>
      </listitem>
      <listitem draggable="true" droppable="true" onDrop="move(event.dragged)">
         <listcell label="(other)"/>
         <listcell label="13,162"/>
```

```
         <listcell label="50.01%"/>
      </listitem>
      <listfoot>
         <listfooter label="Total 132"/>
         <listfooter label="26,267"/>
         <listfooter label="100.00%"/>
      </listfoot>
   </listbox>
   <zscript>
   void move(Component dragged) {
      self.parent.insertBefore(dragged, self);
   }
   </zscript>
</window>
```

### Multiple Types of Draggable Components

It is common that a droppable component doesn't accept all draggable components. For example, an e-mail folder accepts only e-mails and it rejects contacts or others. You could silently ignore non-acceptable components or alert an message, when `onDrop` is invoked.

To have better visual effect, you could identify each type of draggable components with an identifier, and then assign the identifier to the `draggable` attribute.

```
<listitem draggable="email"/>
...
<listitem draggable="contact"/>
```

Then, you could specify a list of identifiers to the `droppable` attribute to limit what can be dropped. For example, the following image accepts only `email` and `contact`.

```
<image src="/img/send.png" droppable="email, contact" onDrop="send(event.dragged)"/>
```

To accept any kind of draggable components, you could specify `"true"` to the `droppable` attribute. For example, the following image accepts any kind of draggable components.

```
<image src="/img/trash.png" droppable="true" onDrop="remove(event.dragged)"/>
```

On the other hand, if the `draggable` attribute is `"true"`, it means the component belongs to anonymous type. Furthermore, only components with the `droppable` attribute assigned to `"true"` could accept it.

## HTML Relevant Components

There are several ways to use HTML components with XUL components in the same ZUML page.

### The `html` Component

The simplest way is to use a XUL component called `html`. The content attribute of a `html` component contains a piece of HTML tags, which will be rendered directly to the browser.

```
<html>
   <attribute name="content"><![CDATA[
      <h4>Hello World!</h4>
      <p>Say hello to the whole world.</p>
   ]]></attribute>
</html>
```

### Mix the HTML and XUL Components

With the XML namespace, developers could mix the use of components from HTML and XUL as depicted as follows.

```
<window title="mix HTML demo" xmlns:h="http://www.w3.org/1999/xhtml">
   <h:table border="1">
      <h:tr>
         <h:td>column 1</h:td>
         <h:td>
            <listbox id="list" mold="select">
               <listitem label="AA"/>
               <listitem label="BB"/>
            </listbox>
         </h:td>
      </h:tr>
   </h:table>
</window>
```


### The `include` Component

The `include` component is used to include the output generated by another servlet. The servlet could be anything including JSF, JSP and even another ZUML page.

```
<window title="include demo" border="normal" width="300px">
   Hello, World!
   <include src="/userguide/misc/includedHello.zul"/>
   <include src="/html/frag.html"/>
</window>
```

Like all other attribute, you could dynamically change the `src` attribute to include the output from a different servlet at the run time.

If the included output is another ZUML, developers are allowed to access components in the included page as if they are part of the containing page.

### Including ZUML Pages

If the `include` component is used to include a ZUML page, the included page will become part of the desktop. However, the included page is not visible until the request is processed completely. In other words, it is visible only in the following events, triggered by user or timer.

The reason is that the include component includes a page as late as the Rendering phase[30]. On the other hand, zscript takes place at the Component Creation phase, and onCreate takes place at the Event Processing Phase. They both execute before the inclusion.

```
<window onCreate="desktop.getPages()"> <!-- the included page not available -->
   <include src="/my.zul"/>
   <zscript>
      desktop.getPages(); //the included page not available yet
   </zscript>
   <button label="Hit" onClick="desktop.getPages()"/>
      <!-- Yes, the included page is available when onClick is received -->
</window>
```

If you want to look into the component of an included page, macro components are usually a better option. Refer to the **Macro Components** section in the **ZK User Interface Markup Language** chapter.

## The `iframe` Component

The `iframe` component uses the HTML IFRAME tag to delegate a portion of the display to another URL. Though the appearance looks similar to the `include` component. The concept and meaning of the `iframe` component is different.

The content included by the `include` component is a fragment of the whole HTML page. Because the content is part of the HTML page, the content is part of the desktop and you could access any components, if any, inside of the `include` component. The inclusion is done at the server, and the browser knows nothing about it. It means the URL specified by the `src` attribute could be any internal resource.

The content of the `iframe` component is loaded by the browser as a separate page. Because it is loaded as a separate page, the format of the content could be different from HTML. For example, you could embed an PDF file.

```
<iframe src="/my.pdf"/>
...other HTML content
```

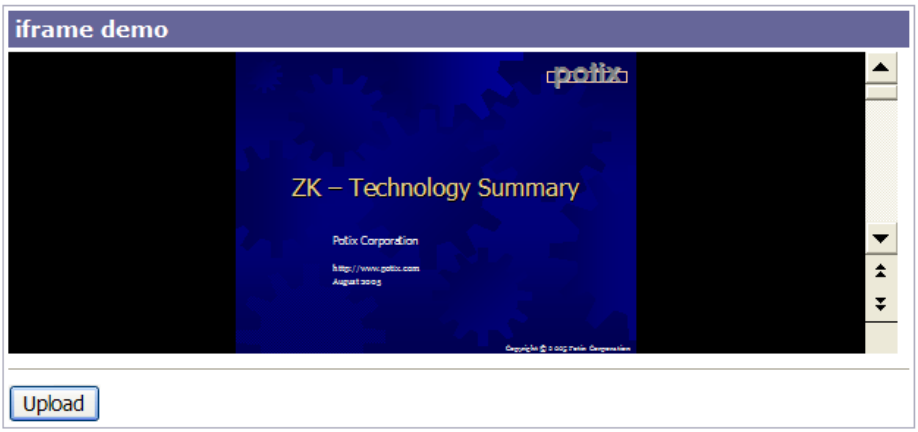
The *embedding* is done by the browser, when it interprets the HTML page containing the IFRAME tag. It also implies that the URL must be a resource that you can access from the browser.

30 Refer to the **Component Lifecycle** chapter for more details.

Like the `image` and `audio` components[31], you could specify the dynamically generated content. A typical example is you could use JasperReport[32] to generate a PDF report in a binary array or stream, and then pass the report to an `iframe` component by wrapping the result with the `com.potix.util.media.AMedia` class.

In the following example, we illustrate that you could embed any content by use of `iframe`, as long as the client supports its format.

```
<window title="iframe demo" border="normal">
   <iframe id="iframe" width="95%"/>
   <separator bar="true"/>
   <button label="Upload">
      <attribute name="onClick">{
         Object media = Fileupload.get();
         if (media != null)
            iframe.setContent(media);
      }</attribute>
   </button>
</window>
```




This picture depicted the appearance after user uploaded an Microsoft PowerPoint file.

## Work with HTML FORM and Java Servlets

The event-driven model is simple and powerful, but it might not be practical to rewrite all servlets to replace with event listeners.

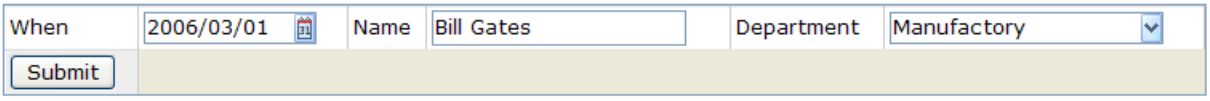
### The `name` Attribute

To work with legacy Web applications, you could specify the name attribute as you did for HTML pages. For example,

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

31 In many ways, `iframe` is much similar to `image` and `audio`. You might consider it as a component for arbitrary content.

32 http://jasperreports.sourceforge.net




```
<h:form method="post" action="/my-old-servlet">
<grid>
  <rows>
<row>When <datebox name="when"/> Name <textbox name="name"/> Department
<combobox name="department">
  <comboitem label="RD"/>
  <comboitem label="Manufactory"/>
  <comboitem label="Logistics"/>
</combobox>
</row>
<row>
  <h:input type="submit" value="Submit"/>
</row>
  </rows>
</grid>
</h:form>
</window>
```

Once users press the submit button, a request is posted to the `my-old-servlet` servlet with the query string as follows.

```
/my-old-servlet?when=2006%2F03%2F01&name=Bill+Gates&department=Manufactory
```

Thus, as long as you maintain the proper associations between name and value, your servlet could work as usual without any modification.

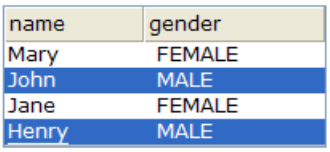
### Components that Support the name Attribute

All input-types components support the name attribute, such as `textbox`, `datebox`, `decimalbox`, `intbox`, `combobox` and `bandbox`.

In additions, list boxes and tree controls are also support the `name` attribute. If the `multiple` attribute is true and users select multiple items, then multiple name/value pairs are posted.

```
<listbox name="who" multiple="true" width="200px">
   <listhead>
      <listheader label="name"/>
      <listheader label="gender"/>
   </listhead>
   <listitem value="mary>
      <listcell label="Mary"/>
      <listcell label="FEMALE"/>
   </listitem>
   <listitem value="john">
      <listcell label="John"/>
      <listcell label="MALE"/>
   </listitem>
   <listitem value="jane">
```

```
      <listcell label="Jane"/>
      <listcell label="FEMALE"/>
   </listitem>
   <listitem value="henry">
      <listcell label="Henry"/>
      <listcell label="MALE"/>
   </listitem>
</listbox>
```




If both John and Henry are selected, then the query string will contain:

```
who=john&who=henry
```

Notice that, to use list boxes and tree controls with the `name` attribute, you have to specify the `value` attribute for `listitem` and `treeitem`, respectively. They are the values being posted to the servlets.

### Rich User Interfaces

Because a `form` component could contain any kind of components, the rich user interfaces could be implemented independent of the existent servlets. For example, you could listen to the `onOpen` event and fulfill a tab panel as illustrated in the previous sections. Yet another example, you could dynamically add more rows to a grid control, where each row might control input boxes with the `name` attribute. Once user submits the form, the most updated content will be posted to the servlet.

## Client Side Actions

Some behaviors are better to be done at the client side with JavaScript codes, such as animations and image rollovers. In order to execute JavaScript codes at the client, ZK introduces the concept of Client Side Actions (CSA). With CSA, developers could listen to any JavaScript event and executes JavaScript codes at the client.

A CSA is similar to an event listener, except an action is is written in JavaScript and executes at the client. ZK allows developers to specify actions for any JavaScript events, such as `onfocus`, `onblur`, `onmouseover` and `onmouseout`, as long as your targeting browsers support them.

The syntax of a client-side action is as follows.

```
action="[focus|blur|mouseover|mouseout|click...]: javascript;"
```

Notice that CSA is totally independent of ZK event listeners, though they might have the same name, such as `onFocus`. The differences include:

- CSA executes at the client side and takes place, before ZK event listener is called at the server.
- CSA codes are written in JavaScript, while ZK event listeners are written in Java.

- CSA could register to any event that your targeting browsers allow, while ZK supports events only list in the **Events** section.

In the following example, we demonstrated how to use client-side actions to provide on-line help. When user change the focus to any of the text boxes, a help message is displayed accordingly.

```
<grid>
   <columns>
      <column/>
      <column/>
      <column/>
   </columns>
   <rows>
      <row>
<label value="text1: "/>
<textbox action="focus: action.show(help1); blur: action.hide(help1)"/>
<label id="help1" visible="false" value="This is help for text1."/>
      </row>
      <row>
<label value="text2: "/>
<textbox action="focus: action.show(help2); blur: action.hide(help2)"/>
<label id="help2" visible="false" value="This is help for text2."/>
      </row>
   </rows>
</grid>
```

text1: This is help for text1.
text2:

## Events

Notice that whether an event is supported depends on a component. In addition, an event is sent after the component's content is updated.

### Mouse Events

| Event Name | Components | Description |
|---|---|---|
| onClick | button image label menuitem toolbutton | Event: com.potix.zk.ui.event.MouseEvent<br>Denotes user has clicked the component. |
| onDblClick | N/A | Event: com.potix.zk.ui.event.MouseEvent<br>Denotes user has double-clicked the component. |

### Keystroke Events

| Event Name | Components | Description |
|---|---|---|
| onOK | window | Event: com.potix.zk.ui.event.KeyEvent<br>Denotes user has pressed the ENTER key. |
| onCancel | window | Event: com.potix.zk.ui.event.KeyEvent<br>Denotes user has pressed the ESC key. |
| onCtrlKey | window | Event: com.potix.zk.ui.event.KeyEvent<br>Denotes user has pressed a key combined with the Ctrl key. |

The keystroke events are sent to the nearest window that has registered an event listener for the specified events. It is designed to implement the submit, cancel and shortcut functions.

As illustrated below, doA() is invoked if user pressed ENTER when T1 got the focus, and doB() is invoked if user pressed ENTER when T2 got the focus.

```
<window id="A" onOK="doA()">
   <window id="B" onOK="doB()">
      <textbox id="T1"/>
   </window>
   <textbox id="T2"/>
</window
```

Notice that a window doesn't receive the keystroke events that are sent for the inner window, unless you post them manually. In the above example, the event won't be sent to window A, if T1 got the focus, no matter whether the onOK handler is declared for window B or not.

### Input Events

| Event Name | Components | Description |
|---|---|---|
| onChange | textbox datebox decimalbox intbox combobox bandbox | Event: com.potix.zk.ui.event.InputEvent<br>Denotes the content of an input component has been modified by the user. |
| onChanging | textbox datebox decimalbox intbox combobox bandbox | Event: com.potix.zk.ui.event.InputEvent<br>Denotes that user is changing the content of an input component. Notice that the component's content (at the server) won't be changed until onChange is received. Thus, you have to invoke the getValue method in the InputEvent class to retrieve the temporary value. |

| Event Name | Components | Description |
|---|---|---|
| onFocus | textbox datebox decimalbox intbox combobox bandbox button toolbarbutton checkbox radio | Event: com.potix.zk.ui.event.Event<br>Denotes when a component gets the focus.<br>Remember event listeners execute at the server, so the focus at the client might be changed when the event listener for onFocus got executed. |
| onBlur | textbox datebox decimalbox intbox combobox bandbox button toolbarbutton checkbox radio | Event: com.potix.zk.ui.event.Event<br>Denotes when a component loses the focus.<br>Remember event listeners execute at the server, so the focus at the client might be changed when the event listener for onBlur got executed. |

**List and Tree Events**

| Event Name | Components | Description |
|---|---|---|
| onSelect | listbox tabbox tree | Event: com.potix.zk.ui.event.SelectEvent<br>Denotes user has selected one or multiple child components. For listbox, it is a set of listitem. For tree, it is a set of treeitem. For tabbox, it is a tab. |
| onOpen | groupbox treeitem combobox bandbox | Event: com.potix.zk.ui.event.OpenEvent<br>Denotes user has opened or closed a component. |

**Slider and Scroll Events**

| Event Name | Components | Description |
|---|---|---|
| onScroll | slider | Event: com.potix.zk.ui.event.ScrollEvent<br>Denotes the content of a scrollable component has been scrolled by the user. |
| onScrolling | slider | Event: com.potix.zk.ui.event.ScrollEvent<br>Denotes that user is scrolling a scrollable component. Notice that the component's content (at the server) won't be changed until onScroll is received. Thus, you have to invoke the getPos method in the ScrollEvent class to retrieve the temporary position. |

**Other Events**

| Event Name | Components | Description |
|---|---|---|
| onCreate | all | Event: com.potix.ui.zk.ui.event.CreateEvent<br>Denotes a component is created when rendering a ZUML page. Refer to the *Component Lifecycle* chapter. |
| onDrop | all | Event: com.potix.ui.zk.ui.event.DropEvent<br>Denotes another component is dropped to the component that receives this event. Refer to the *Drag and Drop* section. |
| onCheck | checkbox radio radiogroup | Event: com.potix.zk.ui.event.CheckEvent<br>Denotes the state of a component has been changed by the user. |
| onMove | window | Event: com.potix.zk.ui.event.MoveEvent<br>Denotes a component has been moved by the user. |
| onShow | window | Event: com.potix.zk.ui.event.ShowEvent<br>Denotes the visibility of a component has been changed by the user. |
| onZIndex | window | Event: com.potix.zk.ui.event.ZIndexEvent<br>Denotes the z-index of a component has been changed by the user. |
| onTimer | timer | Event: com.potix.zk.ui.event.Event<br>Denotes the timer you specified has triggered an event. To know which timer, invoke the getTarget method in the Event class. |
| onNotify | *any* | Event: com.potix.zk.ui.event.Event<br>Denotes a application-dependent event. Its meaning depends on applications. Currently, no component will send this event. |

**The Event Flow of radio and radiogroup**

For developer's convenience, the onCheck event is sent to raido first and then to radiogroup[33]. Thus, you could add listener either to the radio group or to each radio button.

```
<radiogroup onCheck="fruit.value = self.selectedItem.label">
   <radio label="Apple"/>
```

33 The internal implementation is done by adding a listener when a radio is added to a radiogroup.

```
   <radio label="Orange"/>
</radiogroup>
You have selected : <label id="fruit"/>
```

The above sample has the same effect as follows.

```
<radiogroup>
   <radio label="Apple" onCheck="fruit.value = self.label"/>
   <radio label="Orange" onCheck="fruit.value = self.label"/>
</radiogroup>
You have selected : <label id="fruit"/>
```

# 8. ZUML with the XHTML Component Set

This chapter describes the set of XHTML components.

## The Goal

The introduction of the XHTML component set is aimed to make it easy to port existent Web pages to ZUML. The ultima goal is that all valid XHTML pages are valid ZUML pages. All servlets handling the submitted form work as usual.

Therefore, existent XHTML pages could share the most powerful advantage that ZUML pages have: rich user interfaces. The richness could be achieved in two ways. First, you could embed Java codes to manipulate XHTML components dynamically. Second, you could add off-of-shelf XUL components into existent pages, just like you add XHTML into XUL pages.

### A XHTML Page Is A Valid ZUML Page

The Web page illustrated below is a simple but typical example.

```
<html>
<head>
<title>ZHTML Demo</title>
</head>
<body>
   <h1>ZHTML Demo</h1>
   <ul id="ul">
      <li>The first item.</li>
      <li>The second item.</li>
   </ul>
   <input type="button" value="Add Item""/>
   <br/>
   <input id="inp0" type="text"/> +
   <input id="inp1" type="text"/> =
   <text id="out"/>
</body>
</html>
```

By naming it with the `zhtml` extension[34], it will be interpreted as a ZUML page by ZK loader. Then, instances of `com.potix.zhtml.Html`, `com.potix.zhtml.Head` and others are created accordingly. In other words, we created a tree of XHTML components at the server. Then, ZK renders them into a regular XHTML page and sends it back to the browser, like what we did for any ZUML pages.

34 If you want every HTML pages to be ZUML pages, you could map the `.html` extension to DHtmlLayoutServlet. Refer to **Appendix A** for details.

### Server-Centric Interactivity

As being a ZUML page, it could embed any Java codes and execute them in the server as follows.

```
<html xmlns:zk="http://www.potix.com/2005/zk">
<head>
<title>ZHTML Demo</title>
</head>
<body>
   <h1>ZHTML Demo</h1>
   <ul id="ul">
      <li>The first item.</li>
      <li>The second item.</li>
   </ul>
   <input type="button" value="Add Item" zk:onClick="addItem()"/>
   <br/>
   <input id="inp0" type="text" zk:onChange="add()"/> +
   <input id="inp1" type="text" zk:onChange="add()"/> =
   <text id="out"/>
   <zscript>
   void addItem() {
      Component li = new Raw("li");
      li.setParent(ul);
      new Text("Item "+ul.getChildren().size()).setParent(li);
   }
   void add() {
      out.setValue(inp0.getValue() + inp1.getValue());
   }
   </zscript>
</body>
</html>
```

In the above example, we use the ZK namespace to specify the `onClick` attribute. It is necessary because XHTML itself has an attribute with the same name.

It is interesting to note that all Java codes are running at the server. Thus, unlike JavaScript you are used to embed in HTML pages, you could access any resource at the server directly. For example, you could open a connection to a database and retrieve the data to fill in certain components.

```
<zscript>
import java.sql.*;
void addItem() {
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   String url = "jdbc:odbc:Fred";
   Connection conn = DriverManager.getConnection(url,"myLogin", "myPassword");
   ...
   conn.close();
}
</zscript>
```

### Servlets Work As Usual

In traditional Web applications, a XHTML page usually submits a form to a specific servlet for processing. You don't need to modify them to port the page to ZK.

## The Differences

Besides being ZK components, the implementation of the XHTML component set has some differences from other component sets[35], such that it would be easier to port traditional XHTML pages to ZK.

### UUID Is ID

Traditional servlets and JavaScript codes usually depend on the id attribute, so UUID of XHTML components are made to be the same as ID. Therefore, developers need not to change their existent codes to adapt ZK, as shown below.

```
<img id="which"/>
<script><!-- JavaScript and running at the browser -->
   function change() {
      var el = document.getElementById("which");
      el.src = "something.gif";
   }
</script>
<zscript><!-- Java and running at the server -->
   void change() {
      which.src = "another.gif";
   }
</zscript>
```

Notice that UUID is immutable and nothing to do with ID for components other than XHTML. Thus, the above example will fail if XUL components are used. If you really want to reference a XUL component in JavaScript, you have to use EL expression to get the correct UUID.

```
<input id="which"/>
<script>
   var el = document.getElementById("${which.uuid}");
</script>
```

### Side Effects

Since UUID is ID, you cannot use the same ID for any two components in the same desktop.

35 These differences are made by implementing particular interfaces, so you could apply similar effects to your own components if you like.

### All Tags Are Valid

Unlike XUL or other component sets, there is no invalid XML element in the XHTML component set. ZK uses the `com.potix.zhtml.Raw` class for constructing any unrecognized XML element[36]. Therefore, developers could use any tags that the target browser supports, no matter whether they are implemented as ZK components.

Similarly, you could use the `Raw` component to create any component not defined in the XHTML component set as follows.

```
new Raw("object"); //object could be any tag name the target browser supports
```

### Case Insensitive

Unlike XUL or other component sets, the component name of XHTML is case-insensitive. The following XML elements are all mapped to the `com.potix.zhtml.Br` component.

```
<br/>
<BR/>
<bR/>
```

### No Mold Support

XHTML components outputs its content directly. They don't support molds. In other words, the mold attribute is ignored.

## The DOM Tree at the Browser

After porting XHMTL pages to ZK, you don't need to manipulate the DOM tree at the browser with JavaScript, though ZK doesn't prevent you from doing that. Rather, you manipulate XHTML components at the server, and then ZK engines updates the DOM tree at the browser for you.

It is convenient but there is a catch. ZK assumes the DOM tree at the browser is the same as the component tree at the server. In most cases, it is true. However, it is not always true.

### The TABLE and TBODY Tags

The browser always creates TBODY between TABLE and TR. Thus, the following two tables have the same structure.

```
<table>
   <tr><td>Hi</td></tr>
</table>
<table>
   <tbody>
      <tr><td>Hi</td></tr>
```

36 Note: this is done by implementing the `com.potix.zk.ui.ext.DynamicTag` interface.

```
   </tbody>
</table>
```

Unfortunately, their component trees are not the same in ZK. Thus, if you want to dynamically manipulate a table, you have to declare TBODY between TABLE and TR. Of course, you don't need to worry this for static tables.

## Events

All XHTML components support the following events, but whether it is applicable still depends on the browsers. For example, `onChange` is meaningless to non-input components, say `body` and `div`. You have to consult the HTML standard[37]

| Event Name | Components | Description |
|---|---|---|
| `onChange` | *all* | Event: `com.potix.zk.ui.event.InputEvent`<br><br>Denotes the content of an input component has been modified by the user. |
| `onClick` | *all* | Event: `com.potix.zk.ui.event.MouseEvent`<br><br>Denotes user has clicked the component. |

## Integrate with JSF, JSP and Others

When integrating with existent Web pages, you might have to ask yourself a few questions.

- Is the existent page static or dynamically generated?
- Is it a minor enhancement, if you want to enrich an existent page? Or, you prefer to rewrite a portion of it?
- Do you prefer to use XUL or XHTML as the default component set when adding a new page?

Depending your requirements, there are several approaches to take.

### Work with Existent Servlets

By use of the form component, you could post a request to an existent servlet. Refer to the **Work with HTML FORM and Java Servlets** section in the **ZUML with the XUL Component Set** chapter for details.

Because the form component might contain any components, you could design rich user interfaces without modifying the existent servlet.

37 http/www.w3c.org

### Enrich by Inclusion

If you prefer to rewrite a portion of an existent page, it might be better to put the rewritten portion in a separate ZUML file. Then, you include the ZUML file from the existent page. For example, you could use jsp:include if JSP technology is used.

```
<jsp:include page="/my/ria.zul"/>
```

### Enrich a Static HTML Page

If you prefer to modify a static HTML page directly by adding the rich content, you could rename it to have the `zhtml` extension. Then, ZK loader is responsible to load the page, and then you could enrich with ZK.

### Enrich a Dynamically Generated Page

If you prefer to modify a dynamically generated HTML page, you could map the DHtmlLayoutFilter to process the generated page. Here is a sample (a part of `web.xml`).

```
<filter>
   <filter-name>zkFilter</filter-name>
   <filter-class>com.potix.zk.ui.http.DHtmlLayoutFilter</filter-class>
   <init-param>
      <param-name>extension</param-name>
      <param-value>html</param-value>
   </init-param>
</filter>
<filter-mapping>
   <filter-name>zkFilter</filter-name>
   <url-pattern>/my/dyna.jsp</url-pattern>
</filter-mapping>
<filter-mapping>
   <filter-name>zkFilter</filter-name>
   <url-pattern>/my/dyna/*</url-pattern>
</filter-mapping>
```

Notice that, if you want to filter the output from include and/or forward, remember to specify the dispatcher element with REQUEST and/or INCLUDE. Consult the Java Servlet Specification for details. For example,

```
<filter-mapping>
   <filter-name>zkFilter</filter-name>
   <url-pattern>/my/dyna/*</url-pattern>
   <dispatcher>REQUEST</dispatcher>
   <dispatcher>INCLUDE</dispatcher>
   <dispatcher>FORWARD</dispatcher>
   <dispatcher>ERROR</dispatcher>
</filter-mapping>
```

### XUL or XHTML

There is no straight answer here. It depends your preferences.

However, a rule of thumb might be whether you want to write the HTML, HEAD and BODY tags to control the overall look of a page. If yes, use XHTML as the default namespace (by naming the file with the `zhtml` extension). If no, use XUL as the default namespace (by naming the file with the `zul` extension).

Remember that you could mix different component sets in the same page by use of the XML namespace to separate them. Moreover, the namespace for the empty prefix is independent of the extension you choose. For example, the following statements are valid no matter what extension you use.

```
<window xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
xmlns:h="http://www.w3.org/1999/xhtml">
   <h:table>
...
```

It is equivalent to the following.

```
<x:window xmlns:x="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
xmlns="http://www.w3.org/1999/xhtml">
   <table>
...
```

# 9. Macro Components

There are two ways to implement a component. One is to implement a class deriving from the `com.potix.zk.ui.AbstractComponent` class. The other is to implement it by use of other components.

The former one is more flexible. It requires deeper understanding of ZK, so it is usually done by component developers. It is discussed in the Component Development Guide.

On the other hand, implementing a new component by use of other components is straightforward. It works like composition, macro expansion, or inline replacement. For sake of convenience, we call this kind of components as *macro components*., while the others are called *native components*.

> **Tip:** a macro component is *no different* from a native component from application developer's viewpoint, except how it is implemented.

## Three Steps to Use Macro Components

It takes three steps to use macro components as follows.

1. Implements a macro component by a ZUML page.
2. Declare the macro component in the page that is going to use it.
3. Use the macro components, which is no difference that other components.

> **Tip:** In addition to define a macro component in page, you can put its definition into a language addon such all pages are able to access the macro component.

### Step 1. The Implementation

All you need to do is to prepare a ZUML page that describes what the component consists of. In other words, the page is a template of the macro.

For example, assume we want to pack a label and a text box as a macro component. Then we could create page, say `/WEB-INF/macros/username.zul`, as follows.

```
<hbox>
   Username: <textbox/>
</hbox>
```

It is done!

The ZUML page implementing a macro component is the same as any other pages, so any ZUML page can be used as a macro component.

### Step 2. The Declaration

Before instantiating a macro component, you have to declare first. One of simplest way to declare is to use the component directives.

```
<?component name="username" macro-uri="/WEB-INF/macros/username.zul"?>
```

As shown, you have to declare the name (the `name` attribute) and the URI of the page (the `macro-uri` attribute).

#### Other Properties

In additions to the `name`, `macro-uri` and `class`[38] attributes, you can specify a list of initial properties that will be used to initialize a component when it is instantiated.

```
<?component name="mycomp" macro-uri="/macros/mycomp.zul"
  myprop="myval" another="anotherval"?>
```

Therefore,

```
<mycomp/>
```

is equivalent to

```
<mycomp myprop="myval1" another="anotherval"/>
```

### Step 3. The Use

The use of a macro component is no different than others.

```
<window>
   <username/>
</window>
```

#### Pass Properties

Like an ordinary component, you can specify properties (aka., attributes) when using a macro component as follows.

```
<?component name="username" macro-uri="/WEB-INF/macros/username.zul"?>
<window>
   <username username="John"/>
</window>
```

All these properties specified are stored in a map that is then passed to the template via a variable called `arg`. Then, in the template, you could access these properties as follows.

```
<hbox>
   Username: <textbox value="${arg.username}"/>
</hbox>
```

38 The class attribute will be discussed later.

**arg.includer**

In additions to the specified properties (aka., attributes), a property called `arg.includer` is always passed with a reference to the macro component being created. It is also the parent component of the components defined in the template. In the above example, `arg.includer` represents the component created by `<username username="John"/>`, and is the parent of `hbox`.

## Macro Components and The ID Space

Like `window`, a macro component is an ID space owner. In other words, it is free to use whatever identifiers to identify components inside the page implementing a macro component (aka., child components of the macro component). They won't conflict with components defined in the same page with the macro component.

For example, assume we have a macro defined as follows.

```
<hbox>
   Username: <textbox id="username"/>
</hbox>
```

Then, the following codes work correctly.

```
<?component name="username" macro-uri="/WEB-INF/macros/username.zul"?>
<zk>
   <username/>
   <button id="username"/> <!-- no conflict because it is in a different ID space -->
</zk>
```

However, the following codes *don't* work.

```
<?component name="username" macro-uri="/WEB-INF/macros/username.zul"?>
<username id="username"/>
```

Why? Like any ID space owner, the macro component itself is in the same ID space with its child components. There are two solutions:

1. Use a special prefix for the identifiers of child components of a macro component. For example, `"mc_username"` instead of `"username"`.

```
<hbox>
   Username: <textbox id="mc_username"/>
</hbox>
```

2. Use the `window` component to create an additional ID space.

```
<window>
   <hbox>
      Username: <textbox id="username"/>
   </hbox>
```

```
</window>
```

The first solution is suggested, if applicable, due to its simplicity.

### Access Child Components From the Outside

Like other ID space owner, you can access its child component by use of two `getFellow` method invocations or `com.potix.zk.ui.Path`.

```
comp.getFellow("username").getFellow("mc_username");
new Path("/username/mc_username");
```

## Provide Additional Methods

Macro components implement the `com.potix.zk.ui.ext.DynamicPropertied` interface, so you can access its properties by use of the `getDynamicProperty` and `setDynamicProperty` methods as follows.

```
<username id="ua" username="John"/>
<button label="what?" onClick="alert(ua.getDynamicProperty(&amp;quot;username&amp;quot;))"/>
```

Obviously, using `DynamicPropertied` is tedious. Worse of all, the visual representation at the client won't be changed if you use `setDynamicProperty` to change a property. Why? First, it only changes the property stored in a macro component. Second, the visual presentation is changed only if some one invokes the `invalidate` method for the macro component (or any its ancestors).

```
ua.setDynamicProperty("username", "Mary"); //visual presentation won't change
ua.invalidate(); //enforce the visual presentation to redraw
```

Thus, it is better to provide a method, say `setUsername`, to manipulate the macro component directly. The following sections will describe two ways to provide additional methods for a macro component.

This can be done by implementing a class and then specify it in the `class` attribute of the component directive, as described in the following sections.

### Provide Additional Methods in Java

It takes two steps to provide additional methods for a macro component.

1. Implement a class by extending from the `com.potix.zk.ui.HtmlMacroComponent` class.

```
//Username.java
package mypack;
public class Username extends HtmlMacroComponent {
   public void setUsername(String name) {
      setDynamicProperty("username", name); //so arg.username will be updated
```

```
      final Textbox tb = (Textbox)getFellow("mc_username");
      if (tb != null) tb.setValue(name); //correct the child if available
   }
}
```

- As depicted above, you have to call `setDynamicProperty` in `setUsername`, because it will be used when the macro component is redrew completely (such as when its parent is redrew).
- Since the `setUsername` method might be called before a macro component creates its children, you have to check whether `mc_username` exists.
- Since `mc_username`'s `setValue` is called, the visual presentation at the client is updated automatically when `setUsername` is called.

2. Declare the class in the macro declaration with the `class` attribute.

```
<?component name="username" macro-uri="/WEB-INF/macros/username.zul"
  class="mypack.Username"?>
```

### Provide Additional Methods in `zscript`

In addition to implementing with a Java file, you can implement the Java class(es) in `zscript`. The advantage is that no compilation is required and you can modify its content dynamically (without re-deploying the Web application). The disadvantage is the performance downgrade and prone to typos.

It takes a few steps to implement a Java class in `zscript`.

1. You have to prepare a zscript file, say `/zs/username.zs`, for the class to implement. Notice that you can put any number of classes and functions in the same `zscript` file.

```
//username.zs
package mypack;
public class Username extends HtmlMacroComponent {
   public void setUsername(String name) {
      setDynamicProperty("mc_username", name);
      ((Textbox)getFellow("mc_username")).setValue(name);
   }
```

2. Use the `init` directive to load the `zscript` file, and then declare the component

```
<?init zscript="/zs/username.zs"?>
<?component name="username" macro-uri="/WEB-INF/macros/username.zul"
  class="mypack.Username"?>
```

The implementation class (`mypack.Username` in the previous example) is resolved as late as the macro component is really used, so it is also OK to use the zscript element to evaluate the `zscript` file.

```
<?component name="username" macro-uri="/WEB-INF/macros/username.zul"
  class="mypack.Username"?>
<zk>
   <zscript src="/zs/username.zs"/>
   <username/>
</zk>
```

Though subjective, the `init` directive is more readable.

### Override the Implementation Class When Instantiation

Like any other component, you can use the `use` attribute to override the class used to implement a macro component for any particular instance.

```
<?component name="username" macro-uri="/WEB-INF/macros/username.zul"
  class="mypack.Username?>

<username use="another.MyAnotherUsername/>
```

Of course, you have to provide the implementation of `another.MyAnohterUsername` in the above example. Once again the class can be implemented with separate Java file, or by use of `zscript`.

### Create a Macro Component Manually

To create a macro component manually, you have to invoke the `postCreate` method after all the initialization as follows.

```
HtmlMacroComponent ua = new HtmlMacroComponent();
ua.setDynamicProperty("username", "Joe");
ua.postCreate(); //then the ZUML page is loaded and child components are created
```

If you implement a class, say Username, for the macro, then you can do as follow.

```
Username ua = new Username();
ua.setUsername("Joe");
ua.postCreate();
```

# 10. Advanced Features

This chapter describes the advance topics about components and pages.

## Identify Pages

All pages in the same desktop could be accessed in an event listener. For the current page of a component, you could use the `getPage` method in the `com.potix.zk.ui.Component` interface.

To get a reference to another page, you first have to assign an identifier to the page being looked for.

```
<?page id="another"?>
...
```

Then, you could use the `getPage` method in the `com.potix.zk.ui.Desktop` interface as follows.

```
<zscript>
   Page another = self.getDesktop().getPage("another");
</zscript>
```

## Identify Components

Components are grouped by the ID spaces. The page itself is an ID space. The `window` component is another ID space. Assume you have a page called P, the page have a window called A, and the window A has a child window B. Then, if you want to retrieve a child component, say C, in the window B. Then, you could do as follows.

```
comp.getDesktop().getPage("P").getFellow("A").getFellow("B").getFellow("C");
```

The `getFellow` method is used to retrieve any fellow in the same ID space. Refer to the **ID Space** section in the **Basics** chapter for the concept of ID spaces.

### The Component Path

Like a path in a file system, a component path is a catenation of IDs of components along ID spaces. In the above example, the path will be "/A/B/C". In other words, the root of a component path is the current page. If you want to identity another page, you have to use "//". In the above example, the path can also be expressed as "//P/A/B/C".

The `com.potix.zk.ui.Path` class is, like `java.io.File`, provided to simplify the manipulation of component paths. Thus, the following statement is equivalent to the above example.

```
Path.getComponent("/A/B/C"); //assume the current page is P
```

```
Path.getComponent("//P/A/B/C");
```

In addition to static methods, you could instantiate a Path instance.

```
Path parent = new Path("//P/A");
new Path(parent, "B/C").getComponent();
```

## Sorting

The list returned from the `getChildren` method of the `com.potix.zk.ui.Component` interface is *live*. So is the `getItems` method of the `com.potix.zul.html.Listbox` interface and others. In other words, you can manipulate it content directly. For example, the following statements are equivalent:

```
comp.getChildren().remove(0);
((Component)comp.getChildren().get(0)).setParent(null);
```

However, you cannot use the `sort` method of the `java.util.Collections` class to sort them. The reason is subtle: the list of children automatically removes a child from the original position, when you add it to another position. For example, the following statement actually moves the second child in front of the first child.

```
comp.getChildren().add(0, comp.getChildren().get(1));
```

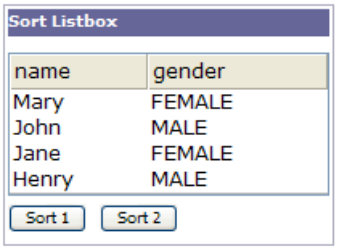
It behaves differently from a normal list (such as LinkedList), so the `sort` method of `Collections` won't work.

To simplify the sorting of components, we therefore provide the `sort` method in the `com.potix.zk.ui.Components` class that works with the list of children.

In the following example, we utilize the `sort` method and the `com.potix.zul.html.ListitemComparator` to provide the sorting for a list box.

Notice that this is only for illustration because list boxes support sorting of list items directly. Refer to the **Sorting** subsection of the **List Boxes** section in the **ZUML with the XUL Component Set** chapter.

```
<window title="Sort Listbox" border="normal" width="200px">
   <vbox>
      <listbox id="l">
         <listhead>
            <listheader label="name"/>
            <listheader label="gender"/>
         </listhead>
         <listitem>
            <listcell label="Mary"/>
            <listcell label="FEMALE"/>
         </listitem>
         <listitem>
```



```
            <listcell label="John"/>
            <listcell label="MALE"/>
         </listitem>
         <listitem>
            <listcell label="Jane"/>
            <listcell label="FEMALE"/>
         </listitem>
         <listitem>
            <listcell label="Henry"/>
            <listcell label="MALE"/>
         </listitem>
      </listbox>
      <hbox>
         <button label="Sort 1" onClick="sort(l, 0)"/>
         <button label="Sort 2" onClick="sort(l, 1)"/>
      </hbox>
   </vbox>
   <zscript>
   void sort(Listbox l, int j) {
      Components.sort(l.getItems(), new ListitemComparator(j));
   }
   </zscript>
</window>
```

## Browser's History Management

In traditional multi-page Web applications, user usually use the BACK and FORWARD button to surf around multiple pages, and bookmark them for later use. With ZK, you still can use multiple pages to represent different set of features and information, as you did in traditional Web applications.

However, it is common for ZK applications to represent a lot of features in one desktop, which usually take multiple Web pages in a traditional Web application. To make user's surfing easier, ZK supports the browser's history management that enables ZK applications to manage browser's history simply in the server.

The concept is simple. You add items for appropriate states of a desktop to the browser's history, and then users can use the BACK and FORWARD button to surf around different states of the same ZK desktop. When users surf around these states, an event called `onBookmarkChanged` is sent to notify the application.

From application's viewpoint, it takes two steps to manage the browser's history:

1. Add an item to the browser's history for each of the appropriate states of your desktop.
2. Listen to the `onBookmarkChanged` event and manipulate the desktop accordingly.

### Add the Appropriate States to Browser's History

Your application has to decide what are the appropriate states to add to the browser's history. For example, in a multi-step operation, each state is a good candidate to add to browser's history, such that users can jump over these states or bookmark them for later use.

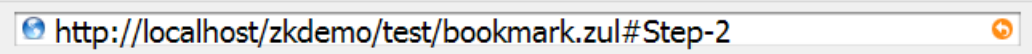
Once you decide when to add a state to the browser's history, you can simply invoke the `setBookmark` method of the `com.potix.zk.ui.Desktop` interface when appropriate. Adding a state to the browser's history is called *bookmarking*. Notice that it is *not* the bookmarks that users add to the browser (aka., My Favorites in Internet Explorer).

> **Tip:** You might call the adding state in the server as the server's bookmarks in contrast with the browser's bookmarks.

For example, assume you want to bookmark the state when the Next button is clicked, then you do as follows.

```
<button label="Next" onClick="desktop.setBookmark(&quot;Step-2&quot;)"/>
```

If you look carefully at the URL, you will find ZK appends `#Step-2` to the URL.




If you press the BACK button, you will see as follows.


### Listen to the `onBookmarkChanged` Event and Manipulate the Desktop Accordingly

After adding a state to the browser's history, users can then surf among these states such as pressing the BACK button to return the previous state. When the state is changed, ZK will notify the application by broadcasting the `onBookmarkChanged` event (an instance of the `com.potix.zk.ui.event.BookmarkEvent` class) to all root components in the desktop.

Unlike traditional multi-page Web applications, you have to manipulate the ZK desktop manually when the state is changed. It is application developer's job to manipulate the desktop to reflect the state that a bookmark represented.

To listen the `onBookmarkChanged` event, you can add an event listener to any page of the desktop, or to any of its root component.

```
<window onBookmarkChanged="goto(event.bookmark)">
   <zscript>
   void goto(String bookmark) {
      if ("Step-2".equals(bookmark)) {
         ...//create components for Step 2
      } else { //empty bookmark
         ...//create components for Step 1
      }
```

```
   </zscript>
</window>
```

Like handling any other events, you can manipulate the desktop as you want, when the `onBookmarkChanged` event is received. A typical approach is to use the `createComponents` method of the `com.potix.zk.ui.Executions` class. In other words, you can represent each state with one ZUML page, and then use `createComponents` to create all components in it when `onBookmarkChanged` is received.

```
if ("Step-2".equals(bookmark)) {
   //1. Remove components, if any, representing the previous state
   try {
      self.getFellow("replacable").detach();
   } catch (ComponentNotFoundException ex) {
      //not created yet
   }

   //2. Creates components belonging to Step 2
   Executions.createComponents("/bk/step2.zul", self, null);
}
```

## Session Timeout Management

After a session is timeout, all desktops it belongs are removed. If a user keeps accessing the desktop that no longer exists, an error message will be shown at the browser to prompt user for the situation.

Sometimes it is better to redirect to another page that gives users more complete description and guides they to the other resources, or asks them to login again. You can specify the target URI, that you want to redirect users to when timeout, in `zk.xml` under `WEB-INF` directory. For example, the target URI is /timeout.zul and then you can add the following lines to `zk.xml`.

```
<session-config>
   <timeout-uri>/timeout.zul</timeout-uri>
</session-config>
```

Tip: For more information about `zk.xml`, refer to **Appendix B**.

In addition to `zk.xml`, you can change the redirect URI programmingly as follows.

```
getWebApp().getConfiguration().setTimeoutURI("/timeout.zul");
```

If you prefer to reload the page instead of redirecting to other URI, you can specify an empty URI as follows.

```
<session-config>
   <timeout-uri></timeout-uri>
</session-config>
```

## Inter-Page Communication

Communications among pages in the same desktop is straightforward. First, you can use event to notify each other. Second, you can use attributes to share data.

### Post and Send Events

You could communicate among pages in the same desktop. The way to communicate is to use the `postEvent` or `sendEvent` to notify a component in the target page.

```
Events.postEvent(new Event("SomethingHappens",
   comp.getDesktop().getPage("another").getFellow("main"));
```

### Attributes

Each component, page, desktop, session and Web application has an independent map of attributes. It is a good place to share data among components, pages, desktops and even sessions.

In `zscript` and EL expressions, you could use the implicit objects: `componentScope`, `pageScope`, `desktopScope`, `sessionScope` and `applicationoScope`.

In a Java class, you could use the attribute-relevant methods in corresponding classes to access them. You could also use the scope argument to identify which scope you want to access. The following two statements are equivalent, assuming comp is a component.

```
comp.getAttribute("some", comp.DESKTOP_SCOPE);
comp.getDesktop().getAttribute("some");
```

## Inter-Web-Application Communication

An EAR file could have multiple WAR files. Each of them is a Web application. There are no standard to communicate between two Web applications.

However, ZK supports a way to reference the resource from another Web applications. For example, assume you want to include a resource, say `/foreign.zul`, from another Web application, say `app2`. Then, you could do as follows.

```
<include src="~app2/foreign.zul"/>
```

Similarly, you could reference a style sheet from another Web application.

```
<?xml-stylesheet href="~app2/foreign.css" type="text/css"?>
```

### Web Resources from Classpath

With ZK, you could reference a resource that is locatable by the classpath. The advantage is that you could embed Web resources in a JAR file, which simplifies the deployment.

```
<img src="~./my/jar.gif"/>
```

Then, it tries to locate the resource, `/my/jar.gif`, at the /web directory by searching resources from the classpath.

# 11. Internationalization

This chapter describes how to make ZK applications flexible enough to run in any locale.

First of all, ZK enables developers to embed Java codes and EL expressions any way you like. You could use any Internationalization method you want, such as java.util.ResourceBundle.

However, ZK has some built-in support of internationalization that you might find them useful.

## Labels

Developers could separate Locale-dependent data from the ZUML pages (and JSP pages) by storing them in i3-label_*lang_CNTY*.properties under the `WEB-INF` directory, where *lang* is the language such as en and fr, and *CNTY* is the country, such as US and FR.

To get a Locale-dependent property, you could use `com.potix.util.resource.Labels` in Java, or `${c:l('key')}` in EL expression. To use it in EL, you have to include the TLD file in your page as follows.

```
<%@ taglib uri="/WEB-INF/tld/web/core.dsp.tld" prefix="c" %>

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

**File Location:** `core.dsp.tld` is distributed under the `dist/WEB-INF` directory.

When a Locale-dependent label is about to retrieved, one of i3-label_*lang_CNTY*.properties will be loaded. For example, if the Locale is `de_DE`, then `WEB-INF/i3-label_de_DE.properties` will be loaded. If no such file, ZK will try to load `WEB-INF/i3-label_de.properties` and `WEB-INF/i3-label.properties` in turn.

In addition, you could extend the label loader to load labels from other locations, say database. Please refer to `com.potix.web.util.resource.impl.LabelLoaderImpl`, which is the default label loader. Also refer to the **The Component Manager** section in the **Beyond ZK** chapter for how to replace the default label with your own implementation.

## Locale-Dependent Files

### Browser and Locale-Dependent URI

Many resources depend on the Locale and, sometimes, the browser that a user is used to visit the Web page. For example, you need to use a larger font for Chinese characters to

have better readability.

ZK can handle this for you automatically, if you specify the URL of the style sheet with "*". The algorithm is as follows.

1. If there is one "*" is specified in an 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.

1. If two or more "*" are specified in an URI such as "/my*/lang*.css", then the first "*" will be replaced with "`ie`" for Internet Explorer and "`moz`" for other browsers[39].
   If the last "*" will be replaced with a proper Locale as described above.

2. All other "*" are ignored.

Notice that the last "*" used to be replaced with a proper Locale must be part of the filename (rather than directory), and it must be right before ".", if any. On the other hand, the first "*" has no such limitation.

In the following examples, we assume the preferred Locale is `de_DE` and the browser is Internet Explorer.

| URI | Resources that are searched |
|---|---|
| /css/norm*.css | 1. /norm_de_DE.css<br>2. /norm_de.css<br>3. /norm.css |
| /css-*/norm*.css | 1. /css-ie/norm_de_DE.css<br>2. /css-ie/norm_de.css<br>3. /css-ie/norm.css |
| /img*/pic*/lang*.png | 1. /imgie/pic*/lang_de_DE.png<br>2. /imgie/pic*/lang_de.png<br>3. /imgie/pic*/lang.png |
| /img*/lang.gif | 1. /img*/lang.gif |

### Locating Browser and Locale Dependent Resources in Java

In additions to component attributes and ZUML attributes, you could handle browser and Locale dependent resource programmingly in Java. Here are a list of methods that you could use.

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

- The `encodeURL`, `forward`, and `include` methods in `com.potix.zk.ui.Exection` for encoding URL, forwarding to another page and including a page. In most cases, these methods are all you need.
- The `locate`, `forward`, and `include` method in `com.potix.web.servlet.Servlets` for locating Web resouces. You rarely need them when developing ZK applications, but useful for writing a servlet, portlet or filter.
- The `encodeURL` method in `com.potix.web.servlet.http.Encodes` for encoding URL. You rarely need them when developing ZK applications, but useful for writing a servlet, portlet or filter.
- The `locate` method in `com.potix.util.resource.Locators` for locating class resources.

## Messages

Messages are stored in properties files which are located at the `/metainfo/mesg` directory of the classpath. Each module is associated with an unique name. In addition, the Locale is appended to the property file, too. For example, the message file of `zk.jar` for Germany messages is `msgzk_de_DN.properties` or `msgzk_de.properties`. Currently, `zk.jar` is only shipped with English and Chinese versions. You could add your own property files for different Locales by placing them at the `/metainfo/mesg` directory of the classpath.

# 12. Database Connectivity

This chapter describes how to make connections to database.

## ZK Is Presentation-Tier Only

ZK is aimed to be as thin as the presentation tier. In additions, with the server-centric approach, it executes all codes at the server, so connecting database is no different from any desktop applications. In other words, ZK doesn't change the way you access the database, no matter you use JDBC or other persistence framework, such as Hibernate[40].

## Simplest Way to Use JDBC (but not recommended)

The simplest way to use JDBC, like any JDBC tutorial might suggest, is to use `java.sql.DriverManager`. Here is an example to store the name and email into a MySQL[41] database.

```
<window title="JDBC demo" border="normal">
   <zscript><![CDATA[
   import java.sql.*;
   void submit() {
      //load driver and get a database connetion
      Class.forName("com.mysql.jdbc.Driver");
      Connection conn = DriverManager.getConnection(
         "jdbc:mysql://localhost/test?user=root&password=my-password");
      PreparedStatement stmt = null;
      try {
         stmt = conn.prepareStatement("INSERT INTO user values(?, ?)");

         //insert what end user entered into database table
         stmt.set(1, name.value);
         stmt.set(2, email.value);

         //execute the statement
         stmt.executeUpdate();
      } finally { //cleanup
         if (stmt != null) {
            try {
               stmt.close();
            } catch (SQLException ex) {
               log.error(ex); //log and ignore
            }
         }
```

40 http://www.hibernate.org

41 http://www.mysql.com

```
         if (conn != null) {
            try {
               conn.close();
            } catch (SQLException ex) {
               log.error(ex); //log and ignore
            }
         }
      }
   }
   </zscript>
   <vbox>
      <hbox>Name : <textbox id="name"/></hbox>
      <hbox>Email: <textbox id="email"/></hbox>
      <button label="submit" onClick="submit()"/>
   </vbox>
</window>
```

Though simple, it is not recommended. After all, ZK applications are Web-based applications, where loading is unpredictable and treasurable resources such as database connections have to be managed effectively.

Luckily, all J2EE frameworks and Web servers support a utility called connection pooling. It is straightforward to use, while managing the database connections well. We will discuss more in the next section.

**Tip:** Unlike other Web applications, it is possible to use `DriverManager` with ZK, though *not recommended*.

First, you could cache the connection in the desktop, reuse it for each event, and close it when the desktop becomes invalid. It works just like traditional Client/Server applications. Like Client/Server applications, it works efficiently only if there are at most tens concurrent users.

To know when a desktop becomes invalid, you have to implement a listener by use of `com.potix.zk.ui.util.DesktopCleanup`.

## Use with Connection Pooling

Connection pooling is a technique of creating and managing a pool of connections that are ready for use by any thread that needs them. Instead of closing a connection immediately, it keeps them in a pool such that the next connect request could be served very efficiently. Connection pooling, in additions, has a lot of benefits, such as control resource usage.

There is no reason not to use connection pooling when developing Web-based applications, including ZK applications.

The concept of using connection pooling is simple: configure, connect and close. The way to connect and close a connection is very similar the ad-hoc approach, while configuration depends on what Web server and database server are in use.

### Connect and Close a Connection

After configuring connection pooling (which will be discussed in the following section), you could use JNDI to retrieve an connection as follows.

```
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;

import javax.naming.InitialContext;
import javax.sql.DataSource;

import com.potix.zul.html.Window;

public class MyWindows extends Window {
   private Textbox name, email;
   public void onCreate() {
      //initial name and email
      name = getFellow("name");
      email = getFellow("email");
   }
   public void onOK() throws Exception {
      DataSource ds = (DataSource)new InitialContext()
            .lookup("java:comp/env/jdbc/MyDB");
         //Assumes your database is configured and
         //named as "java:comp/env/jdbc/MyDB"

      Connection conn = null;
      Statement stmt = null;
      try {
         conn = ds.getConnection();
         stmt = conn.prepareStatement("INSERT INTO user values(?, ?)");

         //insert what end user entered into database table
         stmt.set(1, name.value);
         stmt.set(2, email.value);

         //execute the statement
         stmt.executeUpdate();
         stmt.close(); stmt = null;
            //optional because the finally clause will close it
            //However, it is a good habit to close it as soon as done, especially
            //you might have to create a lot of statement to complete a job
      } finally { //cleanup
         if (stmt != null) {
            try {
               stmt.close();
            } catch (SQLException ex) {
               //(optional log and) ignore
            }
         }
         if (conn != null) {
```

```
            try {
               conn.close();
            } catch (SQLException ex) {
               //(optional log and) ignore
            }
         }
      }
   }
}
```

**Notes:**

- It is important to close the statement and connection after use.
- You could access multiple database at the same time by use of multiple connections. Depending on the configuration and J2EE/Web servers, these connections could even form a distributed transaction.

### Configure Connection Pooling

The configuration of connection pooling varies from one J2EE/Web/Database server to another. Here we illustrated some of them. You have to consult the document of the server you are using.

#### Tomcat 5.5 + MySQL

To configure connection pooling for Tomcat 5.5, you have to edit `$TOMCAT_DIR/conf/context.xml`[42], and add the following content under the `<Context>` element. The information that depends on your installation and usually need to be changed is marked in the blue color.

```
<!-- The name you used above, must match _exactly_ here!
   The connection pool will be bound into JNDI with the name
   "java:/comp/env/jdbc/MyDB"
-->
<Resource name="jdbc/MyDB" username="someuser" password="somepass"
   url="jdbc:mysql://localhost:3306/test"
   auth="Container" defaultAutoCommit="false"
   driverClassName="com.mysql.jdbc.Driver" maxActive="20"
   timeBetweenEvictionRunsMillis="60000"
   type="javax.sql.DataSource" />
</ResourceParams>
```

Then, in `web.xml`, you have to add the following content under the `<web-app>` element as follows.

42 Thanks Thomas Muller (http://asconet.org:8000/antville/oberinspector) for correction.
See also http://tomcat.apache.org/tomcat-5.5-doc/jndi-resources-howto.html and http://en.wikibooks.org/wiki/ZK/How-Tos/HowToHandleHibernateSessions#Working_with_the_Hibernate_session for more details.

```
<resource-ref>
  <res-ref-name>jdbc/MyDB</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>
```

**JBoss + MySQL**

The following instructions is based on section 23.3.4.3 of the reference manual of MySQL 5.0.

To configure connection pooling for JBoss, you have to add a new file to the directory called deploy (`$JBOSS_DIR/server/default/deploy`). The file name must end with "`-ds.xml`", which tells JBoss to deploy this file as JDBC Datasource. The file must have the following contents. The information that depends on your installation and usually need to be changed is marked in the blue color.

```
<datasources>
    <local-tx-datasource>
        <!-- This connection pool will be bound into JNDI with the name
             "java:/MyDB" -->
        <jndi-name>MyDB</jndi-name>
        <connection-url>jdbc:mysql://localhost:3306/test</connection-url>
        <driver-class>com.mysql.jdbc.Driver</driver-class>
        <user-name>someser</user-name>
        <password>somepass</password>

        <min-pool-size>5</min-pool-size>

        <!-- Don't set this any higher than max_connections on your
         MySQL server, usually this should be a 10 or a few 10's
         of connections, not hundreds or thousands -->

        <max-pool-size>20</max-pool-size>

        <!-- Don't allow connections to hang out idle too long,
         never longer than what wait_timeout is set to on the
         server...A few minutes is usually okay here,
         it depends on your application
         and how much spikey load it will see -->

        <idle-timeout-minutes>5</idle-timeout-minutes>

        <!-- If you're using Connector/J 3.1.8 or newer, you can use
             our implementation of these to increase the robustness
             of the connection pool. -->

        <exception-sorter-class-
name>com.mysql.jdbc.integration.jboss.ExtendedMysqlExceptionSorter</exception-
sorter-class-name>
```

```
        <valid-connection-checker-class-
name>com.mysql.jdbc.integration.jboss.MysqlValidConnectionChecker</valid-
connection-checker-class-name>

    </local-tx-datasource>
</datasources>
```

**JBoss + PostgreSQL**

```
<datasources>
   <local-tx-datasource>
   <!-- This connection pool will be bound into JNDI with the name
        "java:/MyDB" -->
   <jndi-name>MyDB</jndi-name>

   <!-- jdbc:postgresql://[servername]:[port]/[database name] -->
   <connection-url>jdbc:postgresql://localhost/test</connection-url>

   <driver-class>org.postgresql.Driver</driver-class>
   <user-name>someuser</user-name>
   <password>somepass</password>
   <min-pool-size>5</min-pool-size>
   <max-pool-size>20</max-pool-size>
   <track-statements>false</track-statements>
   </local-tx-datasource>
</datasources>
```

## ZK Features Applicable to Database Access

### The `com.potix.zk.ui.event.EventThreadCleanup` Interface

As emphasized before, it is important to close the connection in the `finally` clause, such that every connection will be returned to connection pool correctly.

To make your application more robust, you could implement the `com.potix.zk.ui.event.EventThreadCleanup` interface to close any pending connections and statements, in case that some of your application codes might forget to close them in the `finally` clause.

However, how to close pending connection and statements really depend on the server you are using. You have to consult the document of the server for how to write one.

> **Tip**: In many cases, it is not necessary (and not easy) to provide such method, because most implementation of connection pooling be recycled a connection if its `finalized` method is called.

### Access Database in EL Expressions

In additions to access database in an event listener, it is common to access database to fulfill an attribute by use of an EL expression. In the following example, we fetch the data from database and represent them with `listbox` by use of EL expressions.

```
<zscript>
   import my.CustomerManager;
   customers = new CustomerManager().findAll(); //load from database
</script>
<listbox id="personList" width="800px" rows="5">
   <listhead>
      <listheader label="Name"/>
      <listheader label="Surname"/>
      <listheader label="Due Amount"/>
   </listhead>
   <listitem value="${each.id}" forEach="${customers}">
      <listcell label="${each.name}"/>
      <listcell label="${each.surname}"/>
      <listcell label="${each.due}"/>
   </listitem>
</listbox>
```

There are several way to implement the `findAll` method.

#### Read all and Copy to a LinkedList

The simplest way is to retrieve all data in the `findAll` method, copy them into a list and then close the connection.

```
public class CustomerManager {
   public List findAll() throws Exception {
      DataSource ds = (DataSource)new InitialContext()
            .lookup("java:comp/env/jdbc/MyDB");

      Connection conn = null;
      Statement stmt = null;
      ResultSet rs = null;
      List results = new LinkedList();
      try {
         conn = ds.getConnection();
         stmt = conn.createStatement();
         rs = stmt.executeQuery("SELECT id, name, surname FROM customers");
         while (rs.next()) {
            long id = rs.getInt("id");
            String name = rs.getString("name");
            String surname = rs.getString("surname");
            results.add(new Custoemr(id, name, surname));
         }
         return results;
      } finally {
```

```
         if (rs != null) try { rs.close(); } catch (SQLException ex) []
         if (stmt != null) try { stmt.close(); } catch (SQLException ex) []
         if (conn != null) try { conn.close(); } catch (SQLException ex) []
      }
   }
}
```

### Implement the `com.potix.zk.ui.util.Initiator` Interface

Instead of mixing Java codes with the view, you could use the `init` Directive to load the data.

```
<?init class="my.AllCustomerFinder" arg0="customers"?>

<listbox id="personList" width="800px" rows="5">
   <listhead>
      <listheader label="Name"/>
      <listheader label="Surname"/>
      <listheader label="Due Amount"/>
   </listhead>
   <listitem value="${each.id}" forEach="${customers}">
      <listcell label="${each.name}"/>
      <listcell label="${each.surname}"/>
      <listcell label="${each.due}"/>
   </listitem>
</listbox>
```

Then, implement the `my.CustomerFindAll` class with the `com.potix.zk.ui.util.Initiator` interface.

```
import com.potix.zk.ui.Page;
import com.potix.zk.ui.util.Initiator;

public class AllCustomerFinder implements Initiator {
   public void doInit(Page page, Object[] args) {
      try {
         page.setVariable((String)args[0], new CustomerManager().findAll());
            //Use setVariable to pass the result back to the page
      } catch (Exception ex) {
         throw UiException.Aide.wrap(ex);
      }
   }
   public void doCatch(Throwable ex) { //ignore
   }
   public void doFinally() { //ignore
   }
}
```

### Transaction and `com.potix.zk.util.Initiator`

For sophisticated application (such as distributed transaction), you might have to control the

lifecyle of a transaction explicitly. If all database access is done in event listeners, there is nothing to change to make it work under ZK. You start, commit or rollback a transaction the same way as suggested in the document of your J2EE/Web server.

However, if you want the evaluation of the whole ZUML page (the Component Creation Phases) is done in the same transaction, then you, as described in the above section, could implement the `com.potix.zk.util.Initiator` interface to control the transaction lifecycle for a given page.

The skeletal implementation is illustrated as follows.

```
import com.potix.zk.ui.Page;
import com.potix.zk.ui.util.Initiator;

public class TransInitiator implements Initiator {
   private boolean _err;
   public void doInit(Page page, Object[] args) {
      startTrans(); //depending the container, see below
   }
   public void doCatch(Throwable ex) {
      _err = true;
      rollbackTrans(); //depending the container, see below
   }
   public void doFinally() {
      if (!_err)
         commitTrans(); //depending the container, see below
   }
}
```

As depicted, the transaction starts in the `doInit` method, and ends in the `doFinally` method of the `com.potix.zk.util.Initiator` interface.

How to start, commit and rollback an transaction depends on the container you use.

#### J2EE Transaction and `Initiator`

If you are using a J2EE container, you could look up the transaction manager (`javax.transaction.TransactionManager`), and then invoke its `begin` method to start an transaction. To rollback, invoke its `rollback` method. To commit, invoke its `commit` method.

#### Web Containers and `Initiator`

If you are using a Web container without transaction managers, you could start a transaction by constructing a database connection. Then, invoke its `commit` and `rollback` methods accordingly.

```
import java.sql.*;
import javax.sql.DataSource;
import javax.naming.InitContext;
```

```
import com.potix.util.logging.Log;
import com.potix.zk.ui.Page;
import com.potix.zk.ui.util.Initiator;

public class TransInitiator implements Initiator {
   private static final Log log = Log.lookup(TransInitiator.class);
   private Connection _conn;
   private boolean _err;

   public void doInit(Page page, Object[] args) {
      try {
         DataSource ds = (DataSource)new InitialContext()
            .lookup("java:comp/env/jdbc/MyDB");
         _conn = ds.getConnection();
      } catch (Throwable ex) {
         throw UiException.Aide.wrap(ex);
      }
   }
   public void doCatch(Throwable t) {
      if (_conn != null) {
         try {
            _err = true;
            _conn.rollback();
         } catch (SQLException ex) {
            log.warning("Unable to roll back", ex);
         }
      }
   }
   public void doFinally() {
      if (_conn != null) {
         try {
            if (!_err)
               _conn.commit();
         } catch (SQLException ex) {
            log.warning("Failed to commit", ex);
         } finally {
            try {
               _conn.close();
            } catch (SQLException ex) {
               log.warning("Unable to close transaction", ex);
            }
         }
      }
   }
}
```

# 13. Portal Integration

ZK provides a portlet to load ZUML pages for JSR 168 compliant portal. This portlet is called ZK portlet loader, and it is implemented as `com.potix.zk.ui.http.DHtmlLayoutPortlet`.

## Configuration

### WEB-INF/portlet.xml

To use it, you first have to add the following definition into `WEB-INF/portlet.xml`. Notice that `expiration-cache` must be set to zero to prevent portals from caching the result.

```
<portlet>
   <description>ZK loader for ZUML pages</description>
   <portlet-name>zkLoader</portlet-name>
   <display-name>ZK Loader</display-name>

   <portlet-class>com.potix.zk.ui.http.DHtmlLayoutPortlet</portlet-class>

   <expiration-cache>0</expiration-cache>

   <supports>
      <mime-type>text/html</mime-type>
      <portlet-mode>VIEW</portlet-mode>
   </supports>

   <supported-locale>en</supported-locale>

   <portlet-info>
      <title>ZK</title>
      <short-title>ZK</short-title>
      <keywords>ZK,ZUML</keywords>
   </portlet-info>
</portlet>
```

### WEB-INF/web.xml

ZK portlet loader actually delegates the loading of ZUML pages to ZK loader (`com.potix.zk.ui.http.DHtmlLayoutServlet`). Thus, you have to configure `WEB-INF/web.xml` as specified in Appendix A, even if you want to use only portlets.

## The Usage

### The zk_page Parameter and Attribute

ZK portlet loader is a generic loader. To load a particular ZUML page, you have to specify either a request parameter or a portlet attribute called `zk_page`. In other words, ZK portlet loader first checks any request parameter called `zk_page`, and then any portlet attribute called `zk_page`. If found, the specified ZUML page is loaded and rendered.

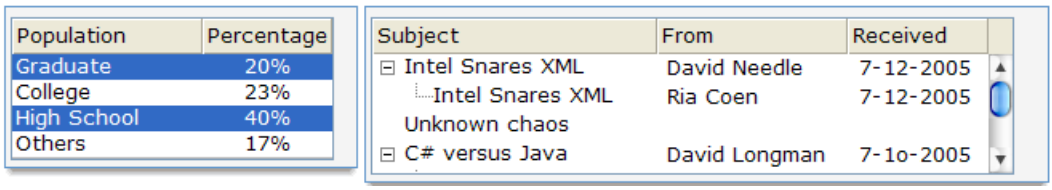
### Examples

How to pass a request parameter or attribute to a portlet depends on the portal. You have to consult the user's guide of your favorite portal for details. The following is an example that uses Potix Portal.

```
<layout contentType="text/html">
   <title>ZK Porlet Demo</title>
   <header name="Cache-Control" value="no-cache"/>
   <header name="Pragma" value="no-cache"/>

   <vbox>
      <hbox>
         <servlet page="sample1.zul"/>
         <portlet name="zkdemo.zkLoader">
            <attribute name="zk_page" value="/test/sample2.zul"/>
         </portlet>
      </hbox>
   </vbox>

   <molds uri="~./pxp/html/molds.xml"/>
</layout>
```



# 14. Beyond ZK

In addition to processing ZUML pages, the ZK distribution included a lot of technologies and tools. This chapter provided the basic information of some of them. Interested readers might look at Javadoc for detailed API.

## Logger

Package: `com.potix.util.logging.Log`

The logger used by ZK is based on the standard logger, `java.util.Logger`. However, we wrap it as `com.potix.util.logging.Log` to make it more efficient. The typical use is as follows.

```
import com.potix.util.logging.Log;
class MyClass {
   private static final Log log = Log.lookup(MyClass.class);
   public void f(Object v) {
      if (log.debugable()) log.debug("Value is "+v);
   }
}
```

Since ZK uses the standard logger to log message, you could control what to log by configuring the logging of the Web server that you are using.

### How to Monitor `i3-log.conf`

In addition to configure the logging of Web server, you could configure ZK to load a file called `i3-log.conf`. If found, it loads its content to initialize the log levels. ZK keeps watching this file and reload its content if the file is modified.

To configure ZK to load and monitor `i3-log.conf`, you have to specify the following content in `WEB-INF/zk.xml`. Refer to **Appendix B** fore more details.

```
<zk>
   <log>
      <log-base>com.potix</log-base>
   </log>
</zk>
```

The above configuration can be done manually by invoking the `init` method of `LogService` as follows.

```
com.potix.util.logging.LogService.init("com.potix");
```

If you want to log not just `com.potix` but everything, you could specify empty `log-base`.

### Content of `i3-log.conf`

An example of `i3-log.conf` is as follows.

```
com.potix.zk.ui.impl.UiEngineImpl=FINER
   #Make the log level of a specified class to FINER
com.potix.zk.ui.http=DEBUG
   #Make the log level of a specified class to DEBUG
com.potix.zk.au.http.DHtmlUpdateServlet=OFF
   #Clear the log level of a specified class such that it inherits what
   #has been defined above (Default: INFO)
com.potix=WARNING
   #Make all leg level of ZK classes to WARNING except those specified here
```

### Location of `i3-log.conf`

At first, ZK looks for this file in the classpath. If not found, it looks for the `conf` directory.

| Application Server | Location |
|---|---|
| Tomcat | Place `i3-log.conf` under the `$TOMCAT_HOME/conf` directory |
| Others | Try the conf directory first. If not working, you could set the system property called the `com.potix.io.conf.dir` directory to be the directory where `i3-log.conf` resides. |

## DSP

Package: `com.potix.web.servlet.dsp`

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

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

## iDOM

Package: `com.potix.idom`

An implementation of W3C DOM. It is inspired by JDOM[43] to have concrete classes for all XML objects, such as Element and Attribute. However, iDOM implements the W3C API, such as org.w3c.dom.Element. Thus, you could use iDOM seamlessly with XML utilities that only accept the W3C DOM.

43 http://www.jdom.org

A typical example is XSLT and XPath. You could use any of favorite XSL processor and XPath utilities with iDOM.

## XAWK

Package: `com.potix.xawk`

Like AWK in Unix, XAWK provided a template-based approach to process XML files. Developers could specify the condition in regular expression (against XPath) and Java codes to evaluate if the condition is satisfied. The Java codes are interpreted at the run time by BeanShell.

```
<rule>
   <pattern>/portlet-app/.*/init-param</pattern><!-- condition -->
   <begin>aname = avalue = null;</begin><!-- Java codes -->
   <end>paramSet.add(aname, avalue);</end><!-- Java codes -->
</rule>
```

## The Component Manager

Package: `com.potix.comp`

> *It is a bit of confusing that what we say component in this section means a generic object, not an UI component.*

The component manager provides a way to let developers override the default behavior, such as the label loader mentioned in the **Internationalization** chapter.

The mechanism is to declare what class you prefer to use in the `i3-comp.xml` file under the jar file's `metainfo` directory. Then, once the jar is loaded, ZK knows which class to use.

```
<component>
   <name>com.potix.util.resource.LabelLoader</name>
   <implementation-
class>com.potix.web.util.resource.impl.LabelLoaderImpl</implementation-class>
</component>
```

# Appendix A. WEB-INF/web.xml

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

## ZK Loader

[Required] Class: `com.potix.zk.ui.http.DHtmlLayoutServlet`

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

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

**The Initial Parameters**

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

## ZK AU Engine

[Required] Class: `com.potix.zk.au.http.DHtmlUpdateServlet`

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

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

## ZK Session Cleaner

[Required] Class: `com.potix.zk.ui.http.HttpSessionListener`

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

## DSP Loader

[Required] Class: com.potix.web.servlet.dsp.InterpreterServlet

InterpreterServlet is a servlet used to load DSP pages upon user's requests. Many ZK components use DSP as templates, so this servlet must be included into `web.xml`. See the **DSP** section in the **Beyond ZK** chapter for details.

## ZK Filter

[Optional] Class: `com.potix.zk.ui.http.DHtmlLayoutFilter`

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

**The Initial Parameters**

| init-param | Descriptions |
|---|---|
| extension | [Optional][Default: `html`]<br>It specifies how to process the response generated by other servlets.<br>If `html` or `zhtml`, XHTML is assumed to be the default namespace. If `xul` or `zul`, XUL is assumed to be the default namespace. |

**How to Specify in web.xml**

```
<filter>
   <filter-name>zkFilter</filter-name>
   <filter-class>com.potix.zk.ui.http.DHtmlLayoutFilter</filter-class>
   <init-param>
      <param-name>extension</param-name>
      <param-value>html</param-value>
      <description>
The extension to identify the content being post-processed by this filter. Default:
html.
      </description>
   </init-param>
</filter>
```

## Sample of web.xml

```
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-
app_2_4.xsd">
   <!-- /// -->
   <!-- DSP -->
   <servlet>
      <description><![CDATA[
The servlet loads the DSP pages.
      ]]></description>
      <servlet-name>dspLoader</servlet-name>
      <servlet-class>com.potix.web.servlet.dsp.InterpreterServlet</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>dspLoader</servlet-name>
      <url-pattern>*.dsp</url-pattern>
   </servlet-mapping>
   <!-- /// -->

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

      <!-- Must. Specifies URI of the update engine (DHtmlUpdateServlet).
      It must be the same as <url-pattern> for the update engine.
      -->
      <init-param>
         <param-name>update-uri</param-name>
         <param-value>/zkau</param-value>
      </init-param>
      <load-on-startup>1</load-on-startup><!-- MUST -->
   </servlet>
   <servlet-mapping>
      <servlet-name>zkLoader</servlet-name>
      <url-pattern>*.zul</url-pattern>
   </servlet-mapping>
   <servlet-mapping>
      <servlet-name>zkLoader</servlet-name>
      <url-pattern>*.zhtml</url-pattern>
   </servlet-mapping>
   <servlet>
```

```
    <description>The asynchronous update engine for ZK</description>
    <servlet-name>auEngine</servlet-name>
    <servlet-class>com.potix.zk.au.http.DHtmlUpdateServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>auEngine</servlet-name>
    <url-pattern>/zkau/*</url-pattern>
  </servlet-mapping>
  <!-- //// -->

  <!-- MIME mapping -->
  <mime-mapping>
    <extension>dsp</extension>
    <mime-type>text/html</mime-type>
  </mime-mapping>
  <mime-mapping>
    <extension>gif</extension>
    <mime-type>image/gif</mime-type>
  </mime-mapping>
  <mime-mapping>
    <extension>html</extension>
    <mime-type>text/html</mime-type>
  </mime-mapping>
  <mime-mapping>
    <extension>htm</extension>
    <mime-type>text/html</mime-type>
  </mime-mapping>
  <mime-mapping>
    <extension>jpeg</extension>
    <mime-type>image/jpeg</mime-type>
  </mime-mapping>
  <mime-mapping>
    <extension>jpg</extension>
    <mime-type>image/jpeg</mime-type>
  </mime-mapping>
  <mime-mapping>
    <extension>js</extension>
    <mime-type>application/x-javascript</mime-type>
  </mime-mapping>
  <mime-mapping>
    <extension>png</extension>
    <mime-type>image/png</mime-type>
  </mime-mapping>
  <mime-mapping>
    <extension>txt</extension>
    <mime-type>text/plain</mime-type>
  </mime-mapping>
  <mime-mapping>
    <extension>xml</extension>
    <mime-type>text/xml</mime-type>
  </mime-mapping>
  <mime-mapping>
```

```
      <extension>zhtml</extension>
      <mime-type>text/html</mime-type>
   </mime-mapping>
   <mime-mapping>
      <extension>zul</extension>
      <mime-type>text/html</mime-type>
   </mime-mapping>

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

# Appendix B. WEB-INF/zk.xml

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

## Overview

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

### The `listener` Element

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

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

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

#### The `com.potix.zk.ui.event.EventThreadInit` Interface

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

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

If a developer wants to prevent an event from being processed, he can throw an exception in the `prepare` method. On the other hand, exceptions thrown by the `init` method are ignored (and only logged).

A typical use of this feature is to implement auto-authentication. For example, JBoss[44] required you to call `SecurityAssociation.setPrincipal` to grant permissions of a user to the event processing thread, as described in the **Initialization Before Processing**

44 http://www.jboss.org

**Each Event** section, the **Event Listening and Processing** chapter.

#### The `com.potix.zk.ui.event.EventThreadCleanup` interface

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

Exceptions thrown by the `init` method are ignored (and only logged).

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

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

#### The `com.potix.zk.ui.util.EventThreadSuspend` interface

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

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

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

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

#### The `com.potix.zk.ui.util.EventThreadResume` interface

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

If a listener implements this interface, an instance is create, and then the `afterResume` method is called, after an event processing thread has been resumed, or aborted.

If an event processing thread is resumed, the `afterResume` method is called in the event processing thread (so it shares the same thread-local storage with `EventThreadSuspend`).

If an event processing thread is aborted (due to, say, cleaning up a desktop), the `afterResume` method is called in other thread (maybe another event processing thread, maybe a cleanup thread).

#### The `com.potix.zk.ui.util.SessionInit` interface

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

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

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

#### The `com.potix.zk.ui.util.SessionCleanup` interface

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

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

#### The `com.potix.zk.ui.util.DesktopInit` interface

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

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

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

#### The `com.potix.zk.ui.util.DesktopCleanup` interface

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

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

#### The `com.potix.zk.ui.util.Monitor` interface

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

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

### The `log` Element

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

```
<log>
   <log-base>com.potix</log-base>
```

```
</log>
```

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

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

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

### The `desktop-config` Element

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

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

#### The `theme-uri` Element

[Default: *none*]

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

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

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

Notice:

1. All style sheets defined in `lang.xml` and `lang-addon.xml` are loaded, no matter this parameter is defined or not. It is convenient for developers to override certain styles.
2. Each JAR could specify a `lang-addon.xml` file (under the `metainfo/zk` directory), so you could specify style sheets there if you have more than one style sheets.
3. You could specify extra CSS files for individual ZUML pages by inserting processing instructions in them as follows.

   ```
   <?xml-stylesheet href="/my.css" type="text/css"?>
   ```

#### The `desktop-timeout` Element

[Default: 3600]

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

#### The `file-check-period` Element

[Default: 5]

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

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

### The `session-config` Element

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

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

#### The `timeout-uri` Element

[Default: *null*]

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

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

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

#### The `session-timeout` Element

[Default: *depending on the Web server*]

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

#### The `max-desktops-per-session` Element

[Default: 10]

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

### The `language-config` Element

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

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

#### The `addon-uri` Element

[Default: *none*]

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

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

### The `system-config` Element

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

```
<system-config>
   <max-event-threads>100</max-event-threads>
   <max-upload-size>5120</max-upload-size>
   <cache-provider-class>my.CacheProvider</cache-provider-class>
   <ui-factory-class>my.UiFactory</ui-factory-class>
   <engine-class>my.UiEngine</engine-class>
</system-config>
```

#### The `max-event-threads` Element

[Default: 100]

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

#### The `max-upload-size` Element

[Default: 5120]

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

#### The `cache-provider-class` Element

[Default: `com.potix.zk.ui.impl.SessionDesktopCacheProvider`]

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

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

Available implementations are as follows.

| Class | Description |
|---|---|
| `com.potix.zk.ui.impl.SessionDesktopCacheProvider` | It stores all desktops from the same session in one single cache. It is simple and fast, but not supporting clustering. |
| `com.potix.zk.ui.impl.GlobalDesktopCacheProvider` | It stores all desktops from the same Web application in one single cache. In other words, it doesn't count on session at all.<br><br>It is useful because some Web server, e.g, BEA WebLogic[45], might be configured to use independent sessions for each request. |

#### The `ui-factory-class` Element

[Default: `com.potix.zk.ui.impl.UiFactoryImpl`]

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

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

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

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

#### The `engine-class` Element

[Default: `com.potix.zk.ui.impl.UiEngineImpl`]

It specifies which class used to implement the UI Engine. The class must have a default

45 http://www.bea.com

constructor (without any argument), and implement the `com.potix.zk.ui.sys.UiEngine` interface.

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

### The `el-config` Element

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

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

#### The `evaluator-class` Element

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

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