ZK Client-side Reference

For ZK 7.0.5

Contents

Articles

ZK Client-side Reference 1
- Introduction 1
  - New to JavaScript 1
  - Object Oriented Programming in JavaScript 2
  - Debugging 9
- General Control 10
  - UI Composing 11
  - Event Listening 15
  - Widget Customization 17
  - JavaScript Packaging 20
  - iZUML 21
- Customization 25
  - Actions and Effects 25
  - Alphafix for IE6 26
  - Drag-and-Drop Effects 27
  - Stackup and Shadow 29
- Component Development 32
  - Components and Widgets 32
  - Server-side 36
    - Property Rendering 36
  - Client-side 39
    - Text Styles and Inner Tags 40
    - Rerender Part of Widget 41
- Notifications 42
  - Widget Events 43
  - DOM Events 47
  - Client Activity Watches 48
- Communication 55
  - AU Requests 56
    - Client-side Firing 56
    - Server-side Processing 58
    - JSON 61
  - AU Responses 63
- Language Definition 64

Samples 64
addon-name 66
component 67
depends 70
device-type 71
extension 71
javascript 72
javascript-module 73
label-template 74
language 75
language-addon 75
language-name 76
library-property 76
macro-template 77
namespace 78
native-template 78
renderer-class 79
stylesheet 79
system-property 80
version 80
zscript 81
Widget Package Descriptor 82
function 83
package 84
script 85
widget 86

References

Article Sources and Contributors 87
Image Sources, Licenses and Contributors 89

ZK Client-side Reference

Documentation:Books/ZK_Client-side_Reference

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

<comment>http://books.zkoss.org/wiki/ZK_Client-side_Reference</comment>

Introduction

ZK Client-side Reference is the reference for client-side programming, including component development.

Client-side programming is optional to application developers. However, if you'd like to have more control of the client, please refer to the General Control and Customization sections. The other sections are more for component development.

If you would like to develop a component, you should read ZK Component Development Essentials first for introduction. Then, you could reference this book if there is an issue.

ZK Client Engine is based on jQuery [1] and you could refer to jQuery Documentation [2] for details.

References

[1] http://jquery.com/
[2] http://docs.jquery.com/Main_Page

New to JavaScript

This section is a quick starting guide for Java Programmers to have quick understanding of JavaScript. You could skip this section if you're already familiar with JavaScript. For a complete list of ZK client-side API, please refer to JavaScript API [1]. For jQuery API, please refer to jQuery's documentation [2].

Here we only discuss Javascript. Javascript is actually easy for Java programmers to learn. The real challenge is the manipulation of DOM and the knowledge of CSS. If you're not familiar with them, please refer to JavaScript Tutorial in w3cschools [3].

Version History

Version	Date	Content

References

[1] http://www.zkoss.org/javadoc/latest/jsdoc/
[2] http://docs.jquery.com/
[3] http://www.w3schools.com/js/default.asp

Object Oriented Programming in JavaScript

JavaScript is not an object-oriented language, but ZK provides some utilities to enable object-oriented programming.

The JavaScript Package

Like Java, ZK's JavaScript classes are grouped into different packages. Similar to Java, the JavaScript code is loaded on demand, but it is loaded on per-package basis rather than per-class (i.e., the whole package is loaded if needed).

The dependence of the packages is defined in the so-called Widget Package Descriptor (aka., WPD). If it is about to load a package, all packages it depends will be loaded too.

Define a Package

A package is usually defined implicitly by the use of a WPD file, such as

```
<package name="zul.grid" language="xul/html" depends="zul.mesh,zul.menu">
      <widget name="Grid"/>
      <widget name="Row"/>
      <widget name="Rows"/>
</package>
```

You rarely need to define it explicitly, but, if you want, you could use zk.$package(_global_.String) [1]. For example,

```
zk.$package('com.foo');
```

Similarly, you could, though rarely needed, import a package by the use of zk.$import(_global_.String) [2].

Notice that, if the package is not loaded yet, zk.$import(_global_.String) [2] won't load the package but returns null.

Load Packages

To force one or multiple packages to load, you could use _global_.Function) zk.load(_global_.String, _global_.Function) [3]. Since ZK loads the packages asynchronously, you cannot access any of the code right after the invocation of _global_.Function) zk.load(_global_.String, _global_.Function) [3]. Rather, you should specify the code in the second argument as a function (Function [4]). For example,

```
zk.load("zul.inp, zul.layout", function () { //load zul.inp and
zul.layout
    new zul.layout.Hlayout({
        children: [new zul.inp.Textbox({value: 'foo'}]
   }); //Correct! zul.inp and zul.layout are both loaded
});
new zul.inp.Textbox({value: 'foo'}); //WRONG! zul.inp not loaded yet
```

Do After Load

If you have some code that should execute when a particular package is loaded, you could use _global_.Function) zk.afterLoad(_global_.String, _global_.Function) [5]. Unlike _global_.Function) zk.load(_global_.String, _global_.Function) [3], it won't force the package(s) to load. Rather, it only registers a function that is called when the specified package(s) is loaded by others.

It is useful to customize the default behavior of widgets, since they might be loaded when your code is running. For example, we could customize SimpleConstraint [6] as follows.

```
zk.afterLoad('zul.inp', function () {
  zu.inp.SimpleConstraint.prototype.validate = function (inp, val) {
    //...customized validation
  };
});
```

Then, the above code can be evaluated even if the `zul.inp` package is not loaded yet.

Depends

If the customization requires a lot of codes and you prefer to put it in a separate package, you could use _global_.String) zk.depends(_global_.String, _global_.String) [7] as follows.

```
zPkg.depends('zul.inp', 'com.foo');
```

which declares the `zul.inp` package depends on the `com.foo` package. In other words, `com.foo` will be loaded when `zul.inp` is loaded.

The JavaScript Class

The root of the class hierarchy is Object [8]. To define a new class, you have to extend from it or one of the deriving classes.

Define a Class

To define a new class, you could use _global_.Map, _global_.Map) zk.$extends(zk.Class, _global_.Map, _global_.Map) [9].

```
zk.$package('com.foo');

com.foo.Location = zk.$extends(zk.Object, {
 x: 0,
 y: 0,
 distance: function (loc) {
  return Math.sqrt(Math.pow(this.x - loc.x, 2) + Math.pow(this.y -
loc.y, 2));
 }
},{
 find: function (name) {
  if (name == 'ZK')
   return new com.foo.Location(10, 10);
  throw 'unknown: "+name;
```

```
 }
})
```

The first argument of _global_.Map, _global_.Map) zk.$extends(zk.Class, _global_.Map, _global_.Map) [9] is the base class to extend from. In this case, we extend from `zk.Object`. The second argument is the (non-static) members of the class. In this case, we define two data members (x and y) and one method (distance).

The third argument defines the static members. In this case we define a static method (find). The third argument is optional. If omitted, it means no static members at all.

Unlike Java, the returned object is the class you defined. You can access it directly, such as o.$instanceof(zk.Widget). In addition, the class object, unlike Java, is not an instance of another class. See more Class [10].

Access Methods of Superclass

To access the superclass's method, you have to use zk.Object...) Object.$super(_global_.String, zk.Object...) [11] or _global_.Array) Object.$supers(_global_.String, _global_.Array) [12].

```
com.foo.ShiftLocation = zk.$extends(com.foo.Location, {
 distance: function (loc) {
   if (loc == null) return 0;
   return this.$super('distance', loc);
 }
});
```

As shown above, $super is a method (inherited from Object [8]) to invoke a method defined in the superclass. The first argument is the method name to invoke, and the rest of the arguments are what to pass to the superclass's method.

Remember that JavaScript doesn't provide method overloading, so there is only one method called distance per class, no matter what signature it might have. So, it is safer (and easier) to pass whatever arguments that it might have to the superclass. It can be done by the use of $supers.

```
distance: function (loc) {
 if (loc == null) return 0;
 return this.$supers('distance', arguments); //pass whatever arguments
the caller applied
}
```

Constructor

Unlike Java, the constructor is always called Object.$init() [13], and it won't invoke the superclass's constructor automatically.

```
com.foo.Location = zk.$extends(zk.Object, {
 $init: function (x, y) {
  this.x = x;
  this.y = y;
 }
});
```

Because the superclass's constructor won't be invoked automatically, you have to invoke it manually as follows.

```
com.foo.ShiftLocation = zk.$extends(com.foo.Location, {
 $init: function (x, y, delta) {
  this.$super('$init', x + delta, y + delta);
 }
});
```

Class Metainfo

The class metainfo is available in the class object, which is returned from _global_.Map, _global_.Map) zk.$extends(zk.Class, _global_.Map, _global_.Map) [9]. With the class object, you can access the static members, examine the class hierarchy and so on.

A class is an instance of Class [10].

$instanceof

To test if an object is an instance of a class, use Object.$instanceof(zk.Class) [14], or Object.isInstance(zk.Object) [15].

```
if (f.$instanceof(com.foo.Location)) {
}
if (com.foo.Location.isInstance(f)) { //the same as above
}
```

$class

Each object has a data member called Object.$class [16], that refers to the class it was instantiated from.

```
var foo = new com.foo.Location();
zk.log(foo.$class == com.foo.Location); //true
```

Unlike Java, you can access all static members by the use of the class, including the derived class.

```
MyClass = zk.$extends(zk.Object, {}, {
 static0: function () {}
});
MyDerive = zk.$extends(zk.MyClass, {}, {
 static1: function () {}
});
MyDerive.static0(); //OK (MyClass.static0)
MyDerive.static1(); //OK
```

However, you cannot access static members via the object.

```
var md = new MyDerive();
md.static0(); //Fail
md.static1(); //Fail
md.$class.static0(); //OK
MyDerive.static0(); //OK
```

isInstance and isAssignableFrom

In addition to static members, each class has two important methods, Object.isInstance(zk.Object) [15] and Object.isAssignableFrom(zk.Class) [17].

```
zk.log(com.foo.Location.isAssignableFrom(com.foo.ShiftLocation)); //true
zk.log(com.foo.Location.isInstance(foo)); //true
```

Naming Conventions

Private and Protected Members

There is no protected or private concept in JavaScript. We suggest to prefix a member with '_' to indicate that it is private or *package*, and postfix a member with '_' to indicate protected. Notice it doesn't prevent the user to call but it helps users not to call something he should not.

```
MyClass = zk.$extends(zk.Object, {
 _data: 23, //private data
 check_: function () { //a protected method
 },
 show: function () { //a public method
 }
});
```

Getter and Setter

Some JavaScript utilities the number of arguments to decide whether it is a getter or a setter.

```
location: function (value) { //not recommended
 if (arguments.length) this.location = value;
 else return value;
}
```

However, it is too easy to get confused (at least, with Java's signature) as the program becomes sophisticated. So it is suggested to follow Java's convention (though JavaScript file is slightly bigger):

```
getLocation: function () {
 return this._location;
},
setLocation: function (value) {
 this._location = value;
}
```

In addition, ZK provides a simple way to declare getter and setters by enclosing them with a special name $define. For example,

```
$define: {
    location: null,
    label: function (val) {
        this.updateDomContent_();
    }
}
```

which defines four methods: getLocation, setLocation, getLabel and setLabel. In addition, setLabel() will invoke the specified function when it is called. For more information, please refer to _global_.Map, _global_.Map) zk.$extends(zk.Class, _global_.Map, _global_.Map) [9].

However, if a property is read-only, you can still declare it without get:

```
distance: function (loc) {
  return Math.sqrt(Math.pow(this.x - loc.x, 2) + Math.pow(this.y -
loc.y, 2));
}
```

Furthermore, if a property is read-only and not dynamic, you can allow users to access it directly:

```
if (widget.type == 'zul.wgt.Div') {
}
```

Beyond Object Oriented Programming

JavaScript itself is a dynamic language. You can add a member dynamically.

Add a Method Dynamically

To add a method to all instances of a given class, add the method to prototype:

```
foo.MyClass = zk.$extends(zk.Object, {
});

foo.MyClass.prototype.myfunc = function (arg) {
 this.something = arg;
};
```

To add a method to a particular instance:

```
var o = new foo.MyClass();
o.myfunc = function (arg) {
 this.doSomething(arg);
};
```

To add a static method:

```
foo.Myclass.myfunc = function () {
 //...
};
```

Interfaces

Not interface supported, but it can be 'simulated' by the use of the function name. For example, if an interface is assumed to have two methods: f and g, the implementation can just requires by invoking them, and any object that with these two methods can be passed to it.

Limitations

- You have to specify this explicitly. Remember it is JavaScript, so the default object is window if you don't.

```
$init: function () {
 $super('$init'); //Wrong! It is equivalent to window.$super('$init')
}
```

- $init won't invoke the superclass's $init automatically. You have to invoke it manually. On the other hand, you can, unlike Java, do whatever you want before calling the superclass's $init.

```
$init: function (widget) {
 //codes are allowed here
 this.$super('$init', widget);
 //more codes if you want
}
```

- Data member defined in the second argument of _global_.Map, _global_.Map) zk.$extends(zk.Class, _global_.Map, _global_.Map) [9] are initialized only once. For example, an empty array is assigned to the definition of MyClass when the class is defined in the following example.

```
MyClass = zk.$extends(zk.Object, {
 data: []
});
```

It means that all instances of MyClass will share the same copy of this array. For example,

```
var a = new MyClass(), b = new MyClass();
a.data.push('abc');
zk.log(b.data.length); //it becomes 1 since a.data and b.data is
actually the same
```

Thus, to assign mutable objects, such as arrays and maps ({}), it is better to assign in the constructor.

```
MyClass = zk.$extends(zk.Object, {
 $init: function () {
  this.data = []; //it is called every time an instance is instantiated
 }
});
```

Version History

Version	Date	Content

References

[1] http://www.zkoss.org/javadoc/latest/jsdoc/_global_/zk.html#$package(_global_.String)
[2] http://www.zkoss.org/javadoc/latest/jsdoc/_global_/zk.html#$import(_global_.String)
[3] http://www.zkoss.org/javadoc/latest/jsdoc/_global_/zk.html#load(_global_.String,
[4] http://www.zkoss.org/javadoc/latest/jsdoc/_global_/Function.html#
[5] http://www.zkoss.org/javadoc/latest/jsdoc/_global_/zk.html#afterLoad(_global_.String,
[6] http://www.zkoss.org/javadoc/latest/jsdoc/zul/inp/SimpleConstraint.html#
[7] http://www.zkoss.org/javadoc/latest/jsdoc/_global_/zk.html#depends(_global_.String,
[8] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Object.html#
[9] http://www.zkoss.org/javadoc/latest/jsdoc/_global_/zk.html#$extends(zk.Class,
[10] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Class.html#
[11] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Object.html#$super(_global_.String,
[12] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Object.html#$supers(_global_.String,
[13] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Object.html#$init()
[14] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Object.html#$instanceof(zk.Class)
[15] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Object.html#isInstance(zk.Object)
[16] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Object.html#$class
[17] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Object.html#isAssignableFrom(zk.Class)

Debugging

Here we discuss how to debug the client-side code. For server side debugging, please consult the IDE manual you use.

Debugger

First, it is suggested to get a debugger for the browser you're working with.

Browser	Debugger
Firefox	Firebug [1]. It is not built-in, so you have to download and install it separately.
Firefox	ZKJet. This is a recommend debugging tool. It is not built-in, so you have to download and install it separately.
Internet Explorer 6	Microsoft script debugger , fiddler2(network inspection)
Internet Explorer 7	Microsoft script debugger , fiddler2(network inspection) ,there's another choice is to use Develope Tool in IE8 with IE7 compatible mode
Internet Explorer 8 , 9	Developer Tools It is built-in and you could start it by pressing F12.

Turn Off Compression and Cache

By default, the JavaScript files (ZK packages) will be compressed and cached, which is hard to step in and debug. You could turn off the compression and the cache of JavaScript files by specifying the following in WEB-INF/zk.xml:

```
<client-config>
    <debug-js>true</debug-js>
</client-config>
<library-property>
    <name>org.zkoss.web.classWebResource.cache</name>
    <value>false</value>
</library-property>
```

Version History

Version	Date	Content

References

[1] http://getfirebug.com/

General Control

If you're an application developer

Though optional, you could have the total control of the client's functionality without the assistance of server-side code. Generally, you don't need to do it. You don't need to know how ZK Client Engine and client-side widgets communicate with the server. Their states are synchronized automatically by ZK and components. However, you could control it if necessary. It is the so-called Server-client fusion.

The rule of thumb that is you should handle most of, if not all, events and manipulate UI at the server, since it is much more productive. Then, you could improve the responsiveness and visual effects, and/or reduce the server loading by handling them at the client, when it is appropriate. Notice that JavaScript is readable by any user, so be careful not to expose sensitive data or business logic when migrating some code from server to client.

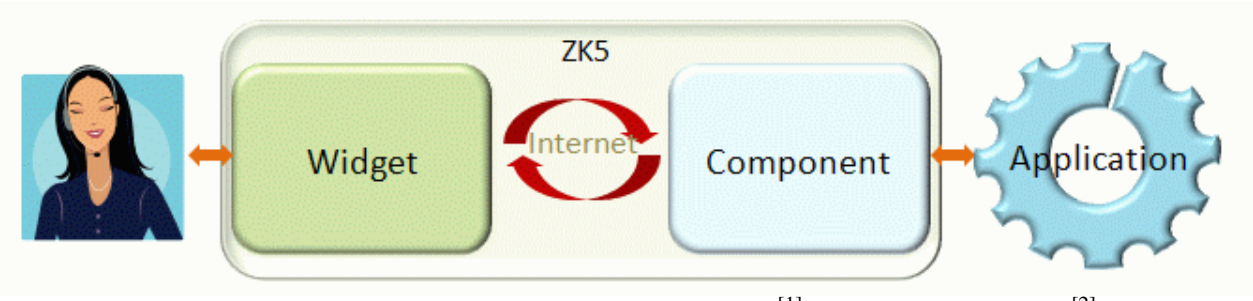
If you're a component developer

This section provides more detailed information about client-side programming, though it is written more for application developers. If you're not familiar with ZK's component development, please refer to ZK Component Development Essentials first.

In this section, we will discuss the details of the client-side control and programming.

UI Composing

Overview



A UI object visible to a user at the client is hosted by a JavaScript object[1] called a widget (Widget [2]). On the other hand, a component is a Java object (Component [3]) representing the UI object at the server that an application manipulates directly. Once a component is attached to a page, a widget is created at the client automatically. Furthermore, any state change of the component at the server will be updated to the widget at the client.

Generally, you need not to know the existence of widgets. Ajax requests and the state synchronization are handled automatically by ZK and the components automatically. However, you could instantiate or alert any client-side widgets directly at the client (in JavaScript). It is the so-called Server+client fusion.

The rule of thumb is that you should handle events and manipulate UI mostly, if not all, at the server, since it is more productive. Then, you could improve the responsiveness and visual effects, and/or reduce the load of the server by handling them at the client, when it is appropriate.

Here we describe how to compose UI in JavaScript at the client.

- For client-side event handling, please refer to the Client-side Event Handling section.
- For XML-based UI composing at the client, please refer to the iZUML section.
- For more information about the relationship among components, widgets and DOM, please refer to the Components and Widgets section]].
- For developing a component, please refer to the Component Development section.

---

[1] It actually depends on the device. For Ajax, it is a JavaScript object. For Android devices, it is a Java object.
[2] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#
[3] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#

Modify Widget's State at Client

While the states of a widget are maintained automatically if you update the corresponding component at the server, you could modify the widget state directly at the server. The modification is straightforward: call the correct method with the arguments you want. Notice that it is JavaScript for Ajax browsers.

```
var foo = zk.Widget.$('$foo');
foo.setValue("What's Up?");
```

For a complete API available to the client-side fusion, please refer to JavaScript API (http:// www. zkoss. org/ javadoc/latest/jsdoc/).

Fusion with Server-side ZUML and Java

It is suggested that the client-side UI composing is better designed to minimize the network round-trip, provide effects and other enhancement, while the most, if not all, of the application is better to be done at the server. Thus, here we only discuss this kind of addon, aka., fusion. For pure-client approach, please refer to Small Talk: ZK 5.0 and Client-centric Approach.

Depending on your requirement, there are typically two situations we could *fuse* the client-side code:

1. Register a client-side event listener.
2. Override widget's default behavior

For example, suppose we want to open the drop down when a commbox gains the focus, then we register a client-side event listener for the onFocus event as follows.

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

As shown, we have to use the client namespace to indicate the onFocus attribute which is for the client-side event listener. It is done by applying XML namespace (http://www.w3schools.com/xml/xml_namespaces.asp):

- Add the xmlns:w="client" attribute
- Prefix w: before onFocus

For more information about the client-side event listener, please refer to the Event Listening section.

The other typical situation to fuse the client-side code is to override the default behavior of a widget. We will discuss it later.

Identify Widget at Client

When the client event is invoked, you can reference the widget using `this` and the event using `event`. In the following example, `this` refers to the label.

```
<window xmlns:w="client">
  <label value="change me by click" w:onClick="this.setValue('clicked');"/>
</window>
```

To retrieve a fellow[1] , you could use Widget.$f(_global_.String) (http://www.zkoss.org/javadoc/latest/jsdoc/zk/ Widget. html#$f(_global_. String)). It works in a similar manner as Component.getFellow(java.lang.String) (http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#getFellow(java.lang.String)). For example,

```
this.$f('foo').setValue('found');
this.$().foo.setValue('found'); //equivalent to the above statement
```

If you don't have a widget as a reference, you could use _global_.Map) Widget.$(zk.Object, _global_.Map) (http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#$(zk.Object,). Notice it assumes there is only one widget with the given ID in all ID spaces of the desktop. For example,

```
zk.Widget.$('foo').setValue('found');
```

In addition, you can use jQuery to select a DOM element of a widget[2] . For example jq("@window") will select DOM elements of all window widgets. And, jq("$win1") will select the DOM elements of all widgets whose ID is win1. (see jq (http://www.zkoss.org/javadoc/latest/jsdoc/_global_/jq.html#)).

```
<window xmlns:w="http://www.zkoss.org/2005/zk/client">
      <vbox>
            <label id="labelone" value="click to change"
                  w:onClick="this.setValue('changed by click label');"
/>

            <button label="button"
                  w:onClick="this.$f('labelone').setValue('changed by
button');" />

            <html><![CDATA[
  <a href="javascript:;" onclick="zk.Widget.$(jq('$labelone')[0]).setValue('changed with
            ]]></html>

      </vbox>
</window>
```

[1] A widget in the same ID space.
[2] Since ZK 5.0.2

Instantiate Widget at Client

A widget has to be created to make a component visible at the client (once it has been attached to a page). However, you could instantiate a widget at client without the corresponding component at the server. To extreme extent, you could create all widgets at the client (of course, this can be costly and less secure).

To instantiate a widget similar to a widget, we can pass all initial values into the constructor. For example,

```
new zul.wnd.Window({
    title: 'Hello, World',
    border: 'normal',
    children: [
        new zul.wgt.Label({value: 'Hi, '}),
        new zul.wgt.Button({
            label: 'Click Me!',
            listeners:  {
                onClick: function (evt) {
                    alert('Hi, you clicked me');
                }
            }
        })
    ]
});
```

As shown, the initial values can be passed as a map. In addition, the children property could be used to specify an array of child widgets, and the listeners property to specify a map of listeners.

In addition to instantiate widgets in JavaScript, you could use a markup language called iZUML. Please refer to the iZUML section for more information.

Attach Widget to DOM

Once a widget is instantiated, you could attach it to the browser's DOM tree to make it visible to users[1] . It can be done in one of two ways:

1. Make it as a child of another widget that already being attached
2. Replace or insert it to a DOM element

You could use Widget.appendChild(zk.Widget) (http:/ / www. zkoss. org/ javadoc/ latest/ jsdoc/ zk/ Widget. html#appendChild(zk.Widget)) or zk.Widget) Widget.insertBefore(zk.Widget, zk.Widget) (http://www.zkoss.org/ javadoc/latest/jsdoc/zk/Widget.html#insertBefore(zk.Widget,). For example,

```
<vlayout>
  <button label="Click Me" xmlns:w="client"
    w:onClick="this.parent.appendChild(new zul.wgt.Label({value:
'Clicked'}))"/>
</vlayout>
```

In addition, we could replace an existent DOM element with a widget (not attached yet). For example,

```
<zk>
  <n:div id="anchor" xmlns:n="native"/>
  <button label="Click Me" xmlns:w="client"
    w:onClick="new zul.wgt.Label({value:
'Clicked'}).replaceHTML('#anchor')"/>
</zk>
```

where we use the native namespace to create a DOM element and then replace it with the label widgt.

---

[1] Notice that a widget is not visible to users unless it is attached to the browser's DOM tree.

When to Run Your JavaScript Code

ZK Client Engine loads a JavaScript package only when it is required. It minimizes the memory footprint at the client. However, this also means that you cannot run your JavaScript code until the required packages have been loaded. It can be done by the use of _global_.Function) zk.load(_global_.String, _global_.Function) (http://www. zkoss.org/javadoc/latest/jsdoc/_global_/zk.html#load(_global_.String,). For example, suppose you're not sure if the zul.wnd and zul.grid package has been loaded, when you are going to instantiate Window (http://www.zkoss. org/javadoc/latest/jsdoc/zul/wnd/Window.html#) and Grid (http://www.zkoss.org/javadoc/latest/jsdoc/zul/ grid/Grid.html#), you could do as follows.

```
zk.load("zul.wnd,zul.grid", function () { //load zul.wnd and zul.grid
if they aren't loaded yet
     //In this function, you could access zul.wnd.Window and
zul.grid.Grid whatever you want
    new zul.wnd.Window({children: [new zul.grid.Grid()]});
});
```

where _global_.Function) zk.load(_global_.String, _global_.Function) (http:/ / www. zkoss. org/ javadoc/ latest/ jsdoc/ _global_/ zk. html#load(_global_. String,) loads the zul.wnd and zul.grid packages and then invokes the function when they have been loaded.

Notice that there is another method for similar purpose called _global_.Function) zk.aferLoad(_global_.String, _global_.Function) (http://www.zkoss.org/javadoc/latest/jsdoc/_global_/zk.html#aferLoad(_global_.String,).

Unlike _global_.Function) zk.load(_global_.String, _global_.Function) (http:// www. zkoss. org/ javadoc/ latest/ jsdoc/ _global_/ zk. html#load(_global_. String,), _global_.Function) zk.afterLoad(_global_.String, _global_.Function) (http://www.zkoss.org/javadoc/latest/jsdoc/_global_/zk.html#afterLoad(_global_.String,) won't load the packages. Rather, it queues the given function and invokes it when the packages have been loaded. It is useful when you want to override the default behavior of a widget. We will discuss it later.

Version History

Version	Date	Content

Event Listening

Overview

ZK allows applications to handle events at both the server and client side. Handling events at the server side, as described in the previous sections, is more common, since the listeners can access the backend services directly. However, handling event at the client side improves the responsiveness. For example, it is better to be done with a client-side listener if you want to open the drop-down list when a comobox gains the focus.

The rule of thumb is to use server-side listeners first since it is easier, and then improve the responsiveness of the critical part, if any, with the client-side listener.

Here we describe how to handle events at the client. For client-side UI manipulation, please refer to the UI Composing and Widget Customization sections.

Declare a Client-side Listener in ZUML

Declaring a client-side listener in a ZUML document is similar to declaring a server-side listener, except

1. Use the client namespace, http://www.zkoss.org/2005/zk/client (aka., client)
2. It is JavaScript
3. Use `this` to reference to the target widget (while the event is referenced with `event`)
4. Use `this.$f()` to reference fellow widgets (Widget.$f() [1])

For example,

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

Notice that EL expressions are allowed in the JavaScript code (for the client-side listener). Thus, it is straightforward to embed the server-side data to the client-side listener. For example,

```
<window id="wnd" title="main">
<combobox xmlns:w="client" w:onFocus="zk.log('${wnd.title}')"/>
</window>
```

If you want to escape it, place a backslash between $ and {, such as w:onFocus="zk.log('$\{wnd.title}')".

For more information about manipulating widgets at the client, please refer to the UI Composing section.

The Relationship between the Client and Server-side Event Listener

It is allowed to register both the client and server-side event listeners. They will be both invoked. Of course, the client-side listener is called first, and then the server-side listener. For example,

```
<div>
  <combobox xmlns:w="client" w:onFocus="this.open()"
   onFocus='self.parent.appendChild(new Label("focus"))'/>
</div>
```

If you want to stop the event propagation such that the server won't receive the event, you could invoke Event.stop(_global_.Map) [2]. For example, the server-side listener won't be invoked in the following example:

```
<div>
  <combobox xmlns:w="client" w:onFocus="this.open(); event.stop();"
   onFocus='self.parent.appendChild(new Label("focus"))'/>
</div>
```

Declare a Client-side Listener in Java

The other way to declare a client-side listener at the server is java.lang.String) Component.setWidgetListener(java.lang.String, java.lang.String) [3]. For example,

```
combobox.setWidgetListener("onFocus", "this.open()");
```

Notice that it is Java and running at the server.

Also notice that EL expressions are not allowed (i.e., not interpreted) if you assign it directly. It is because EL expressions are interpreted by ZK Loader when loading a ZUL page. However, it is easy to construct a string to any content you want with Java.

Register a Client-side Listener in Client-Side JavaScript

Listening an event at the client could be done by calling int) Widget.listen(_global_.Map, int) [4]. For example,

```
<window>
      <bandbox id="bb"/>
      <script defer="true">
      this.$f().bb.listen({onFocus: function () {this.open();}});
      </script>
</window>
```

where

1. defer="true" is required such that the JavaScript code will be evaluated after all widgets are created successfully. Otherwise, it is not able to retreive the bandbox (bb).
2. script is a widget (unlike zscript), so this references to the script widget, rather than the parent.
3. Widget.$f(_global_.String) [5] is equivalent to Component.getFellow(java.lang.String) [6], except it is a JavaScript method (accessible at the client).

Register DOM-level Event Listener

Notice that the event listener handling discussed in the previous sections is for handling so-called ZK widget event (Event [7]). Though rare, you could register a DOM-level event too by the use of jQuery (API: jq [8]).

Version History

Version	Date	Content

References

[1] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#$f()
[2] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Event.html#stop(_global_.Map)
[3] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#setWidgetListener(java.lang.String,
[4] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#listen(_global_.Map,
[5] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#$f(_global_.String)
[6] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#getFellow(java.lang.String)
[7] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Event.html#
[8] http://www.zkoss.org/javadoc/latest/jsdoc/_global_/jq.html#

Widget Customization

Override Widget's Default Behavior

There are many ways to override the default behavior of widgets or even ZK Client Engine. JavaScript is a dynamic language and you could override almost any methods you want.

Override a Widget Method

For example, suppose we want to change the CSS style of a label when its value is changed, then we might have the code as follows.

```
<window xmlns:w="http://www.zkoss.org/2005/zk/client">
      <label>
            <attribute w:name="setValue">
            function (value) {
                  this.$setValue(value); //call the original method
                  if (this.desktop) {
                        this._flag = !this._flag;
                        this.setStyle('background:'+(this._flag ?
'red':'green'));
                  }
            }
            </attribute>
      </label>
</window>
```

where

- We specify client namespace to the setValue attribute to indicate it is the method to override

- The content of the attribute is a complete function definition of the method, including function ()
- You can access the widget by this in the function
- You can access the original method by this.$xxx, where xxx is the method name being overridden. If the method doesn't exist, it is null.
- To retrieve another widget, use this.$f('anotherWidgetId') or other methods as described in the previous section
- You can specify EL expressions[1] in the content of the attribute, such as

```
w:setValue='function (value) { this.$setValue(value + "${whatever}")}';
```

Notice that EL expressions are evaluated at the server before sending back to the client. Thus, you could use any Java class or variables in EL expressions.

[1] EL expressions are allowed since ZK 5.0.2

Override a Default Widget Method

In previous section, we showed how to override the method of a particular widget we declared. This, however, only affects the behavior of a particular instance. If you want to modify the behavior of all instances of a widget class, you have to override the method in prototype[1] .

For example,

```
<window xmlns:w="http://www.zkoss.org/2005/zk/client">
      <label id="labelone" value="label one"/>
      <label id="labeltwo" value="label two"/>
      <script defer="true">
            var oldSV = zul.wgt.Label.prototype.setValue;
            zul.wgt.Label.prototype.setValue = function (){
                  arguments[0]="modified prototype"+arguments[0];
                  oldSV.apply(this, arguments);
            }
      </script>
      <button label="change" onClick="labelone.setValue((new Date()).toString());
    labeltwo.setValue((new Date()).toString());"/>
</window>
```

where we assign a new method to zul.wgt.Label.prototype.setValue. Since it is prototype, the setValue method of all instances are modified.

[1] For more information about JavaScript's prototype, please refer to Using Prototype Property in JavaScript (http://www.packtpub.com/article/using-prototype-property-in-javascript) and JavaScript prototype Property (http://www.w3schools.com/jsref/jsref_prototype_math.asp)

Override a Widget Field

You can override a method or a field no matter it exists or not. For example, it is easy to pass the application-specific data to the client, such as

```
<label value="hello" w:myval="'${param.foo}'"/>
```

Notice that the content of the attribute must be a valid JavaScript snippet. To specify a string (as shown above), you have to enclose it with ' or " if you want to pass a string. It also means you can pass anything, such as new Date().

Override a Widget Method in Java

In addition to ZUML, you could override a Widget method or field by the use of java.lang.String) Component.setWidgetOverride(java.lang.String, java.lang.String) (http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#setWidgetOverride(java.lang.String,) at the server. For example,

```
label.setWidgetOverride("setValue",
    "function (value) {this.$setValue('overloaded setValue');}");
```

Specify Your Own Widget Class

You could specify your own implementation instead of the default widget class (at the client) as follows.

```
<zk xmlns:w="http://www.zkoss.org/2005/zk/client">
  ...
  <button w:use="foo.MyButton"/>
</zk>
```

where foo.MyButton is a widget you implement. For example,

```
zk.afterLoad("zul.wgt", function () {
  zk.$package("foo").MyButton = zk.$extends(zul.wgt.Button, {
    setLabel: function (label) {
      this.$supers("setLabel", arguments);
     //do whatever you want
    }
  });
});
```

Notice that _global_.Function) zk.afterLoad(_global_.String, _global_.Function) (http://www.zkoss.org/javadoc/latest/jsdoc/_global_/zk.html#afterLoad(_global_.String,) is used to defer the declaration of foo.MyButton until zul.wgt has been loaded.

Load Additional JavaScript Files

You could use Script (http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Script.html#), HTML SCRIPT tag or script to load additional JavaScript files. Please refer to script for more information.

The Client-Attribute Namespace

```
[since 5.0.3]
```

The client-attribute namespace (http://www.zkoss.org/2005/zk/client/attribute; shortcut, client/attribute) is used to specify additional DOM attributes that are not generated by widgets. In other words, whatever attributes you specify with the client-attribute namespace will be generated directly to the browser's DOM tree. Whether it is meaningful, it is really up to the browser -- ZK does not handle or filter it at all.

For example, you want to listen to the onload event, and then you can do as follows[1] .

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

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

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

The other use of the client-attribute namespace is to specify attributes that are available only to certain browsers, such as accessibility and Section 508 (http:/ / www. section508. gov/ index. cfm?FuseAction=Content&ID=12#Web).

---

[1] Fore more information, please refer to ZK Component Reference: iframe

Version History

Version	Date	Content

JavaScript Packaging

If you'd like to customize the client-side behavior, it will end up with some JavaScript code. The code can be packaged in several ways depending on the size and re-usability.

It is recommended to take a look at the Object-Oriented Programming in JavaScript section, if you are not familiar how ZK extends JavaScript to support the concept of packages and classes.

Embed the JavaScript Code Directly

Use the script directive to embed the code directly. For example,

```
<!-- foo.zul -->
<?script type="text/javascript" content="jq.IE6_ALPHAFIX='.png';"?>
```

Alternatively, you could use the script component to embed the code.

Put in a Separate File and Reference it in the ZUML page

If there are a lot of JavaScript code, it is better to package them in a separate file, and then use the script directive to reference the file in every ZUML page that requires it.

```
<!-- foo.zul -->
<?script type="text/javascript" src="/myjs/foo.js"?>
```

Put in a Separate File and Reference it in Language Addon

If the JavaScript code will be used in every ZUML page, it is better to package them in a separate file, and then make it part of the the language definition. To make it part of the language definition, you could specify the following content in the language addon, say, `WEB-INF/lang-addon.xml`:

```
<language-addon>
    <addon-name>my.extension</addon-name><!-- any name you like -->
```

```
    <javascript src="/myjs/foo.js"/> <!-- assume you package it as /myjs/foo.js -->
</language-addon>
```

Then, you could specify the language addon in `WEB-INF/zk.xml`:

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

Make It a WPD File for More Control

Technically, you could do whatever you want with a JavaScript file. However, if you prefer to make it a JavaScript package, such that they will be loaded automatically when required, you could package them as a WPD file.

For example, you could have a WPD file and make it loaded with the zk package (so it speeds up the loading).

```
<language-addon>
    <addon-name>my.extension</addon-name><!-- any name you like -->
    <javascript package="my.foo" merge="true"/> <!-- assume you call it my.foo -->
</language-addon>
```

Version History

Version	Date	Content

iZUML

Overview

```
ZK 6: Available in all editions
ZK 5: Available in ZK PE and EE only [1]
```

iZUML is a client-side version of ZUML. iZUML is interpreted at the client. You could embed and use iZUML in any Web page, including a ZUML document and pure HTML pages. But, for the sake of description, here we only illustrate the use in pure HTML pages. With some modification, readers could apply it to ZUML document, JSP and other technologies too.

For composing UI in JavaScript at the client, please refer to the Client-side UI Composing section.

How to Embed iZUML into HTML

It is typically placed inside an HTML DIV tag, though you can choose other kind of tags. Furthermore, it is suggested to enclose the iZUML document with an HTML comment (&lt!-- and -->) such that the page can be interpreted correctly by the browser. For example,

```
<div id="main" display="none">
      <!--
      <window>
            What's your name? <textbox onOK="sayHello(this)"/>
```

```
        </window>
        <separator bar="true"/>
        -->
</div>
```

Of course, you construct an iZUML document dynamically such as the follows.

```
var zuml = '<window title="' + title +'">What\'s your name? <textbox/></window>';
```

Specify ZK JavaScript Files

Before using iZUML or other client-side feature, ZK's JavaScript files must be specified (and loaded). If you are using with ZUML, they are loaded automatically. However, if you are using so-called pure-client approach (such as a static HTML file), you have to specify them explicitly as follows.

```
<script type="text/javascript" src="/zkdemo/zkau/web/js/zk.wpd" charset="UTF-8">
</script>
<script type="text/javascript" src="/zkdemo/zkau/web/js/zk.zuml.wpd" charset="UTF-8">
</script>
```

Notice that it is not required if you are using with a ZUML document.

How to Create Widgets from iZUML

If the iZUML document is embedded in a HTML tag, you can use _global_.Map, _global_.Map, _global_.Function) Parser.createAt(_global_.String, _global_.Map, _global_.Map, _global_.Function) [2] to create widgets. If you construct the iZUML document as a string, you could use _global_.String, _global_.Map, _global_.Function) Parser.create(zk.Widget, _global_.String, _global_.Map, _global_.Function) [3].

EL Expression

The EL expression supported by iZUML is actually a JavaScript snippet. It could be any valid JavaScript expression. Unlike ZUML, iZUML's EL expressions start with #{ and end with } [4] . Because the starting character of EL expressions is different, it is easier to embed iZUML's EL expression in a ZUML page.

Here is a list of built-in variables (aka., implicit objects) that you can access in the EL expressions.

table name

Name	Description

this	It references that a widget has been created.
	If the EL expression is part of the if and unless attribute, this is the parent widget. If the EL expression is part of other attribute, this is the widget being created.
	`<window title="some">`
	`  #{this.getTitle()}...`
	`  <texbox if="#{this.border}"/>`
	`</window>`
	where this refers to the window widget in both EL expressions.
	`<window title="some">`
	`  <textbox value="#{this.parent.getTitle()}">`
	`</window>`
	where this refers to the textbox widget.

–	The context passed to the argument named var of _global_.Map, _global_.Map, _global_.Function) Parser.createAt(_global_.String, _global_.Map, _global_.Map, _global_.Function) [2] and _global_.String, _global_.Map, _global_.Function) Parser.create(zk.Widget, _global_.String, _global_.Map, _global_.Function) [3]. `#{_.info.name}`

[1] http://www.zkoss.org/product/edition.dsp
[2] http://www.zkoss.org/javadoc/latest/jsdoc/zk/zuml/Parser.html#createAt(_global_.String,
[3] http://www.zkoss.org/javadoc/latest/jsdoc/zk/zuml/Parser.html#create(zk.Widget,
[4] For 5.0.7 and older version, iZUML's expressions start with `${`. Since 5.0.8, `#{` is recommended, thought `${` is still valid (for backward compatibility) but deprecated.

Built-in Attributes

forEach

```
<window title="Test" border="normal">
       <label forEach="#{[this.getTitle(), this.getBorder()}"/>
</window>
```

- Notice
  - Unlike widget attributes, this references to the parent widget, which is window in the above case.

if

```
<button label="Edit" if="#{_.login}/>
```

unless

```
<button label="View" unless="#{_.inEditMode}"/>
```

Built-in Element

attribute

```
<button label="Click">
      <attribute name="onClick">
```

```
      this.parent.setTitle("Click");
      </attribute>
</button>
```

is equivalent to

```
<button label="Click onClick="this.parent.setTitle('click')"/>
```

zk

The zk element doesn't represent a widget.

```
<zk forEach="#{[1, 2, 3]}">
      #{each} <textbox value="#{each}"/>
</zk>
```

Notes

script

When <script> is specified in iZUML, it actually refers to the script widget (Script (http:// www. zkoss. org/javadoc/latest/jsdoc/zul/utl/Script.html#)) (rather than HTML SCRIPT tag).

style

When <style> is specified in iZUML, it actually refers to the style widget (Style (http://www.zkoss.org/javadoc/latest/jsdoc/zul/utl/Style.html#)) (rather than HTML STYLE tag).

Example

For a more complete example, please refer to Small Talk: ZK 5.0 and Client-centric Approach.

Version History

Version	Date	Content
5.0.8	August 2011	The starting character of iZUML's EL expressions is changed to `#{`, so it is easier to embed iZUML in a ZUML page. For backward compatibility, `${` is still valid but deprecated.
6.0.0	October 2011	iZUML is available to all editions, including CE, PE and EE.

Customization

This section describes the customizable features of ZK Client Engine, and how to customize them. For infomration about packing JavaScript code, please refer to the JavaScript Packaging section.

Actions and Effects

```
[since 5.0.6]
```

Here we describe how to provide more effects for client-side actions.

The allowed effects are actually the names of methods defined in Actions [1]. Thus, to add a new effect, you have to add a new method to it. For example,

```
zk.eff.Actions.fooIn = function (n, opts) {
    //your own effect to make the node visible, such as
    //zk(n).slideIn(this, opts);
};
```

Then, you could use it in the client-side action:

```
<div action="show: fooIn">
....
</div>
```

The signature of an effect method is as follows.

```
function (DOMElement [2] n, Map [3] opts);
```

where `n` is the DOM element to apply the action, and `opts` is the options specified in the client-side action.

Notice that, before invoking jQuery's effects, you should invoke _global_.Map, _global_.Array, boolean) jqzk.defaultAnimaOpts(zk.Widget, _global_.Map, _global_.Array, boolean) [4] to prepare the initial options for animation. For example,

```
this.defaultAnimaOpts(wgt, opts, prop, true).jq
      .css(css).show().animate(anima, { //the rest depending the jQuery
 effect you use
            queue: false, easing: opts.easing, duration: opts.duration
|| 400,
            complete: opts.afterAnima
      });
```

Version History

Version	Date	Content
5.0.6	December 2010	This feature was introduced in 5.0.6

References

[1] http://www.zkoss.org/javadoc/latest/jsdoc/zk/eff/Actions.html#
[2] http://www.zkoss.org/javadoc/latest/jsdoc/_global_/DOMElement.html#
[3] http://www.zkoss.org/javadoc/latest/jsdoc/_global_/Map.html#
[4] http://www.zkoss.org/javadoc/latest/jsdoc/_global_/jqzk.html#defaultAnimaOpts(zk.Widget,

Alphafix for IE6

IE6 failed to render a PNG with alpha transparency correctly. Please refer to here [1] for more information.

ZK provides the fix, but you have to turn it on by specifying a JavaScript variable called jq.IE6_ALPHAFIX [2] For example,

```
<?script content="jq.IE6_ALPHAFIX='.png';"?>
<zk>
  <button image="foo.png"/>
</zk>
```

where `.png` causes all PNG images to be fixed. If you want to fix certain images, you can do as follows

```
<?script content="jq.IE6_ALPHAFIX='more.png|-trans.png'?>
```

If <?script?> doesn't work, you can try using a regular script-component:

```
<zk>
  <script type="text/javascript">jq.IE6_ALPHAFIX='.png';</script>
  <button image="foo.png" />
</zk>
```

If you prefer to use plain Java instead of ZUL files, you can instantiate a Script component and append it to another component:

```
Script alphafix = new Script();
alphafix.setContent("jq.IE6_ALPHAFIX='.png';");
parent.appendChild(alphafix);
```

Version History

Version	Date	Content

References

[1] http://homepage.ntlworld.com/bobosola/index.htm
[2] http://www.zkoss.org/javadoc/latest/jsdoc/_global_/jq.html#IE6_ALPHAFIX

Drag-and-Drop Effects

There are two levels to customize the drag-and-drop effects: per-widget and browser-level.

Per-Widget Customization

Widget [2] has a set of methods for handling drag-and-drop. You could customize them based on your requirement.

If you want to customize a particular widget, you could do as follows[1] .

```
var superwgt = {};
zk.override(wgt, superwgt, {
    initDrag_: function () {
        //your implementation
        superwgt.initDrag_.apply(this, arguments); //if you want to
call back the default implementation
    }
});
```

If you want to override all widgets of particular class, say, Combobox, you could do as follows.

```
var supercomobox = {};
zk.override(zul.inp.Combobox.prototype, supercomobox, {
    initDrag_: function () {
        //your implementation
        supercomobox.initDrag_.apply(this, arguments); //if you want to
 call back the default implementation
    }
});
```

If you override Widget [2], then all widgets are affected[2] .

Here is a list of methods you could override. For a complete list, please refer to Widget [2].

Method	Description
Widget.dropEffect_(boolean) [3]	Called to have some visual effect when the user is dragging a widget over this widget and this widget is droppable. Notice it is the effect to indicate that a widget is droppable.
zk.Event) Widget.onDrop_(zk.Draggable, zk.Event) [4]	Called to fire the onDrop event. You could override it to implement some effects to indicate dropping.
Widget.getDragOptions_(_global_.Map) [5]	Returns the options used to instantiate Draggable [6]. There is a lot what you could customize with this method, since the options control many effects, such `starteffect`, `endeffect`, `change` and so on. Please refer to Draggable [6] and the source code for more information.
global.Offset) Widget.cloneDrag_(zk.Draggable, _global_.Offset) [7]	Called to create the visual effect representing what is being dragged. In other words, it creates the DOM element that will be moved with the mouse pointer when the user is dragging.
Widget.uncloneDrag_(zk.Draggable) [8]	Undo the visual effect created by _global_.Offset) Widget.cloneDrag_(zk.Draggable, _global_.Offset) [7]. In other words, it removes the DOM element that was created.

[1] _global_.Map, _global_.Map) zk.override(java.lang.Object, _global_.Map, _global_.Map) (http://www.zkoss.org/javadoc/latest/jsdoc/_global_/zk.html#override(java.lang.Object,) is a utility to simplify the overriding of a method.
[2] It is also a browser-level customization
[3] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#dropEffect_(boolean)
[4] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#onDrop_(zk.Draggable,
[5] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#getDragOptions_(_global_.Map)
[6] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Draggable.html#
[7] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#cloneDrag_(zk.Draggable,
[8] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#uncloneDrag_(zk.Draggable)

Browser-level Customization

DnD (http:/ / www. zkoss. org/ javadoc/ latest/ jsdoc/ zk/ DnD. html#) provides a collection of drag-and-drop utilities. By customizing it, all widgets in the same browser will be affected.

For example, if you would like customize *ghosting" of the DOM element being dragged, you could override _global_.Offset, _global_.String) DnD.ghost(zk.Draggable, _global_.Offset, _global_.String)* (http:/ / www. zkoss. org/javadoc/latest/jsdoc/zk/DnD.html#ghost(zk.Draggable,) *as follows.*

```
var superghost = zk.DnD.ghost;
zk.DnD.ghost = function (drag, ofs, msg) {
    if (msg != null)
        return superghost(drag, ofs, msg);
   //do whatever you want
}
```

Version History

Version	Date	Content

Stackup and Shadow

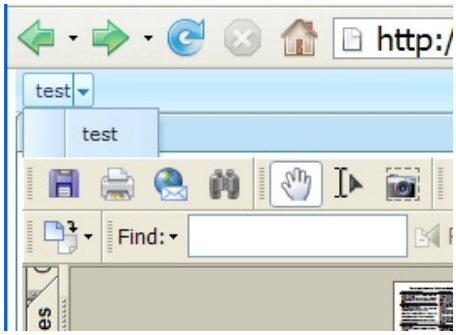
Overview

`zk.useStackup` is a JavaScript variable to indicate how to handle the so-called *stackup* and *autohide* techniques. It is used to resolve the z-index issue when a page contains, say, a PDF iframe.

For example, the following codes will cause the menupopup obscured by the iframe.

```
<zk>
<menubar width="100%">
      <menu label="test">
      <menupopup>
      <menuitem label="test"/>
      <menuitem label="test"/>
      </menupopup>
      </menu>
</menubar>
<window title="iframe/pdf" border="normal" width="500px" sizable="true">
      <iframe style="background-color:transparent" src="/test2/B1896797.pdf" width="100%"
</window>
</zk>
```

For better performance, neither stackup nor autohide is applied by default, so the menupopup is obscured by the iframe as shown below.



Usage

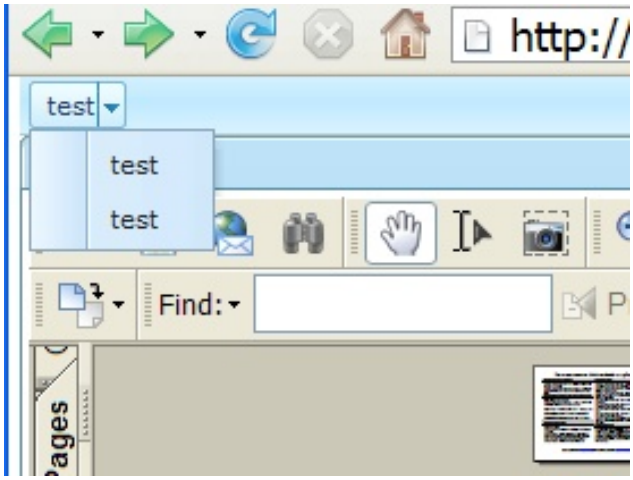
To resolve this, you could specify `'auto'`[1] to `zk.useStackup` as follows.

```
<?script content="zk.useStackup='auto'"?>
<zk>
<menubar width="100%">
...
</zk>
```

In addition, you have to specify `true` to the `autohide` property of the `iframe` component as following.

```
<iframe style="background-color:transparent" src="/test2/B1896797.pdf" width="100%"
autohide="true"/>
```

Notice that no all `iframe` will cause the obscure issue. For example, it is OK if `iframe` contains another HTML page. Thus, specify it only if necessary (for better performance). Here is the correct result.

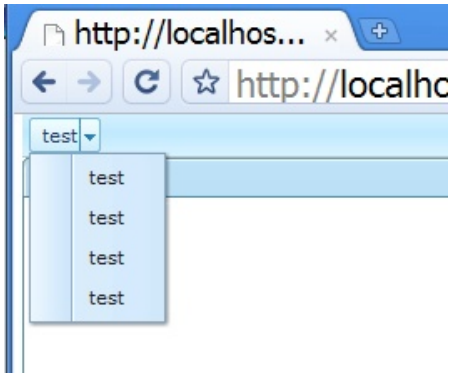


[1] Available since ZK 5.0. For prior version, specify `true` instead.

The stackup and autohide techniques

The stackup technique resolves the obscure issue by placing a transparent iframe under the widget (`menupopup` in this example) that should appear on top. Unfortunately, this technique can not be applied to all browsers. Thus, there is another technique called autohide.

The autohide technique resolves the obscure issue by hiding the iframe that obscures the widget on top. In other words, the iframe is not visible when the menupopup is shown up (on top of the iframe).



All Possible Values

auto

```
[Since 5.0]
```

This is the most common value that can be assigned to `zk.useStackup`. If it is assigned, the stackup technique is applied to Internet Explorer and Firefox, while the `autohide` technique is applied to Safari, Chrome and Opera.

auto/gecko

```
[Since 5.0]
```

Firefox has a problem to show a PDF iframe if two or more iframes are overlapped. In other words, we have to apply the `autohide` technique instead of the `stackup` technique. For example,

```
<?script content="zk.useStackup = 'auto/gecko';"?>
<zk>
<window title="iframe/pdf" border="normal" width="500px" mode="overlapped">
      <iframe style="background-color:transparent" src="/test2/B1896797.pdf"
      width="100%" autohide="true"/>
</window>
<window title="iframe/pdf" border="normal" width="500px" mode="overlapped">
      <iframe style="background-color:transparent" src="/test2/B1896797.pdf"
      width="100%" autohide="true"/>
</window>
</zk>
```

shadow and stackup

When the stackup technique is enabled, a stackup is created for each shadow (of an overlapped window) such that the window appears on top of the others. You can turn off the stackup for an individual window by disabling the shadow property (shadow="false").

For example, if a page has only one overlapped iframe that might contain PDF, you can still use the stackup (instead of autohide, which is slower) by specifying `shadow` as `fasle`, and `zk.useStackup` as `'auto'` (instead of `'auto/gecko'`).

true

```
[Since 3.6.2]
```

Always use the stackup technique.

false

```
[Since 3.6.2]
```

Never use the stackup technique.

Version History

Version	Date	Content

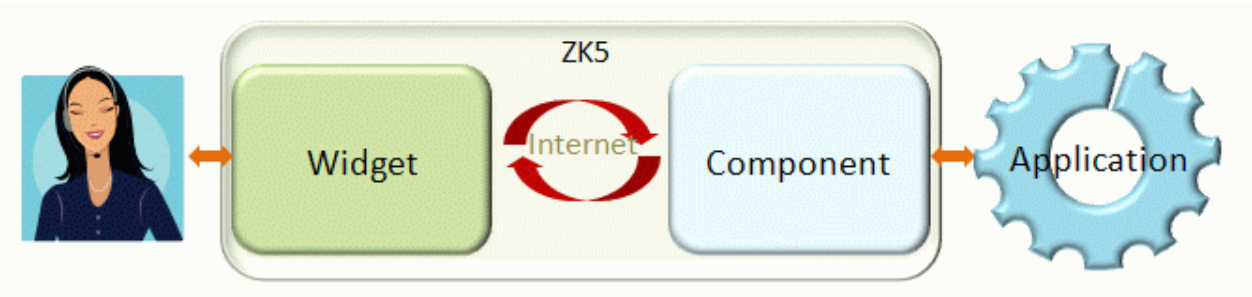
Component Development

This section describes how to develop a component. It has two parts: server-side and client-side.

See Also

- For introduction, please refer to ZK Component Development Essentials.
- For information about the communication between the client and server, please refer to the Communication section.

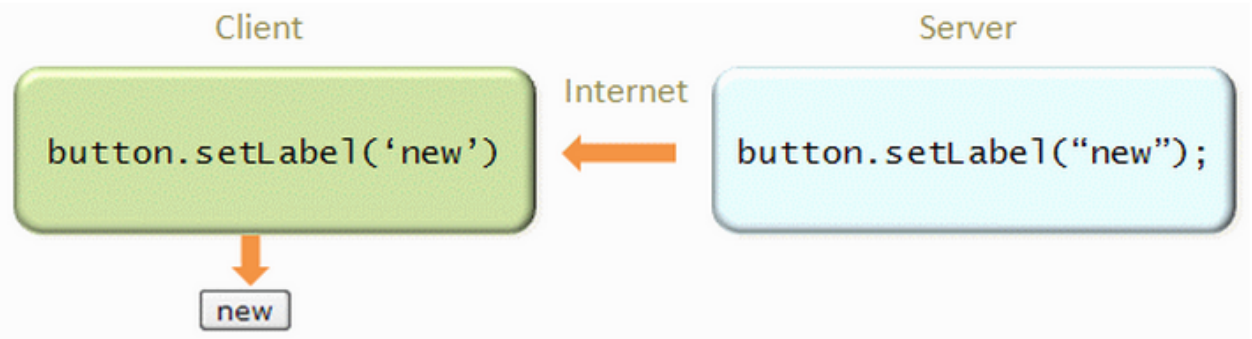
Components and Widgets



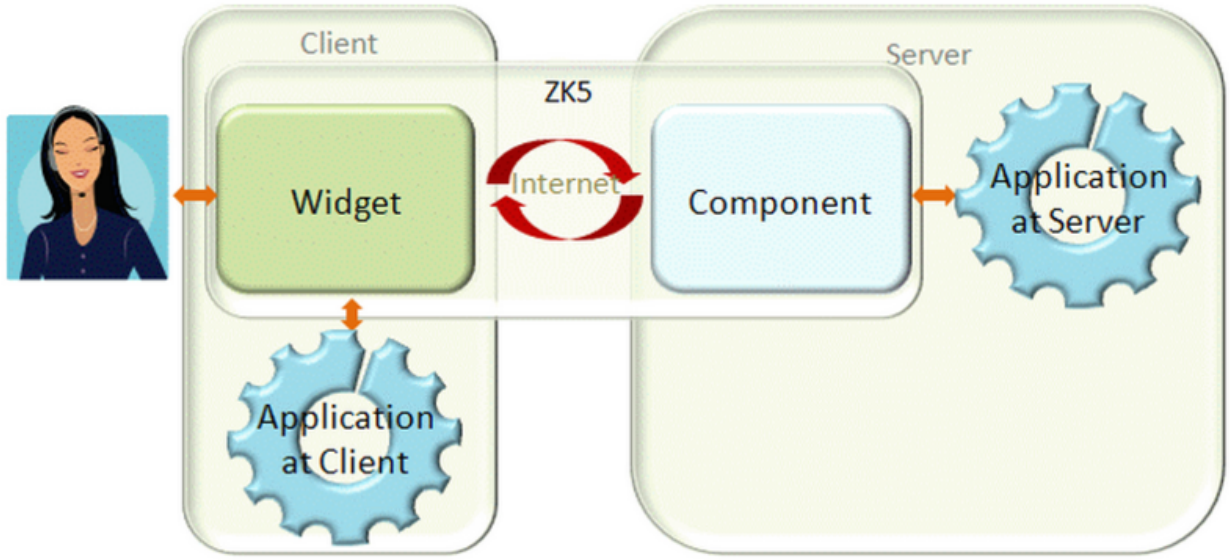
There are two kind of UI objects: components and widget. A component is a Java object running at the server, representing an UI object that an application can manipulate. A component has all the behavior that an UI object should have, except it doesn't have the visual part. For example, the following code snippet creates a window and a textbox.

```
Window w = new Window();
w.setTitle();
w.appendChild(new Textbox());
w.setPage(page); //assuming page is the current page
```

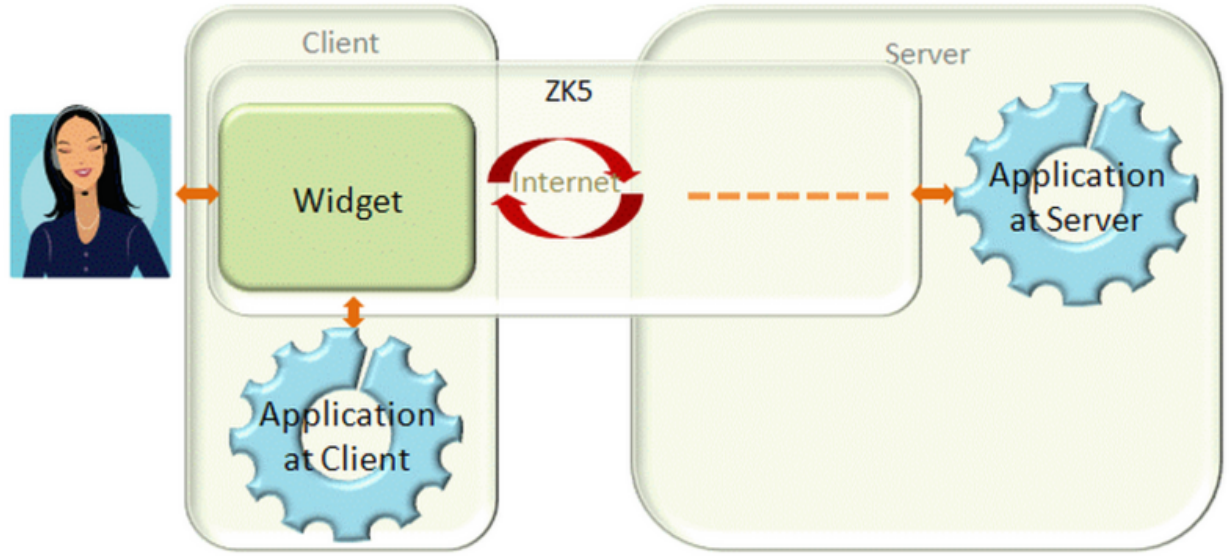
On the other hand, a widget is a JavaScript object running at the client, representing an UI object to interact with the user. To interact with the user, a widget has a visual appearance and handle events happening at the client.



A component and a widget work hand-in-hand to deliver UI to an user and to notify the application about a user's activity, such as clicking and dragging. For example, when an application invokes Button.setLabel(java.lang.String) [1] to change the label of a button component, the Button.setLabel(_global_.String) [2] of corresponding button widget (aka., peer widget) will be invoked at the client to change the visual appearance, as shown right. When the user clicks the button widget, the onClick event will be sent back to the server and notify the application.



Thought not required, a widget is usually implemented with most functions of a component. That means developers can control them directly at the client, as shown left. It improves the responsiveness and reduces the network traffics. Of course, it also increases the development cost. For example, an application might hide or change the order of columns of a grid at the client, while the application running at the server handle the reloading of the whole content of the grid.



Furthermore, a developer can create a widget at the client, and the widget will not have any peer component at the server as shown right. For example,

```
//JavaScript
var w = new zk.wgt.Button();
w.setLabel('OK');
wnd.appendChild(w); //assume wnd is another widget
```

Component and Page

The peer widget of a component is created automatically, when it is attached to a page. On the other hand, if a component is not attached, the client won't know its existence.

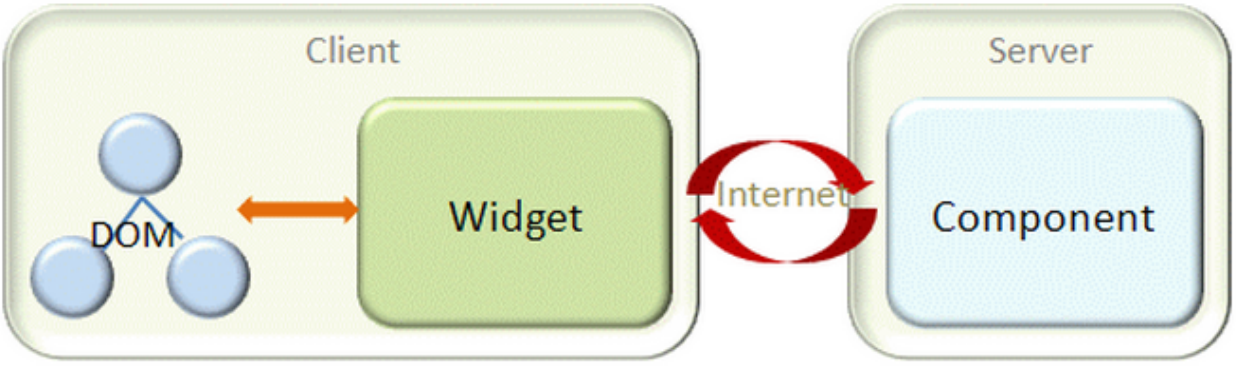
	Server	Client	Description
1	`Window w = new Window();` `w.setTitle("Hello Window");`	*nothing*	A Window component is instantiated but it doesn't have the peer widget. Furthermore, it will be garbage-collected if there is no reference to it

2	`w.setPage(page);`	Auto invoked by ZK Client Engine
		`var pw = new zul.wnd.Window(uuid);`
		`pw.setTitle('Hello World');`
		`pw.replaceHTML(uuid);`
			Attach the component to the specified page, and a peer widget will be created automatically at the client later (after processing the AU Requests).

3	`w.setTitle("Hi ZK");`	Auto invoked by ZK Client Engine `pw.setTitle('Hi ZK');`	Once a component is attached to a page, any following modification will be sent to the client and invoke the corresponding method of the peer widget.

- Notes:
  - There are two ways to attach a component to page. First, call the setPage method to make it as a root component of the page. Second, make a component as a child of another component that is attached to a page. For example, with w.appendChild(new Label("Hi"));, the label will be attached to a page if w is attached.
  - The AU request is a HTTP request, so it is a request-process-response protocol. That is, all client invocation (auto create a widget and so on) will take place after the processing is done and the response is sent back to client.

Widget and DOM



A widget is an UI object at the client. Like Swing's component, creating a widget doesn't make it visible to the user. Rather, you have to attach it to the DOM tree (of the browser).

	Client Widget	Client DOM	Description
1	`var wp = new zul.wnd.Window(); wp.setTitle('Hello World');` Invoked by ZK Client Engine or client application	*nothing*	A window widget is instantiated. If it is called by ZK Client Engine (due to the invocation at the server), it has a peer component. If it is called by client application, there is no peer component.
2	`wp.replaceHTML(uuid);`	Create one or a tree of DOM elements (depending on the implementation of a widget) and replace the specified node.	Attach a widget to the DOM tree, and the visual appearance is shown up (unless it is invisible).
3	`wp.setTitle('Hi ZK');`	Update the DOM element(s) created in the previous step	A modification of the widget will modify the corresponding DOM elements

Attach a widget to the DOM tree

There are several ways to attach a widget to the DOM tree

1. Invoke zk.Desktop, zk.Skipper) Widget.replaceHTML(zk.Object, zk.Desktop, zk.Skipper) [3] to replace an existent DOM element with the DOM element(s) of the widget (aka., the DOM content).
2. Invoke Widget.appendChild(zk.Widget) [4] or zk.Widget) Widget.insertBefore(zk.Widget, zk.Widget) [5] to make a widget a child of another widget that are already attached to the DOM tree.

Version History

Version	Date	Content

References

[1] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Button.html#setLabel(java.lang.String)
[2] http://www.zkoss.org/javadoc/latest/jsdoc/zul/wgt/Button.html#setLabel(_global_.String)
[3] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#replaceHTML(zk.Object,
[4] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#appendChild(zk.Widget)
[5] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#insertBefore(zk.Widget,

Server-side

This section describes how to develop the component at the server-side (Component [3]).

See Also

- For introduction, please refer to ZK Component Development Essentials.
- For the client-side widget development, please refer to the Client-side section.
- For information about the communication between the client and server, please refer to the Communication section.

Property Rendering

If a state (aka., a property) of a component will cause the peer widget to have a different behavior or visual appearance, the state has to be sent to the widget to ensure the consistency.

There are two situations a component has to send the states to the client.

1. Render All Properties When Attached

   A component has to render all properties when it is attached to a page at the first time
2. Dynamic Update a Property

   A component has to send the new value of a property when it is changed dynamically.

> Notice that this section describes how to synchronize states of a component to the widget. To synchronize states back to a component, refer to the AU Requests section.

Render All Properties When Attached

When ZK is about to render a new-attached component to the client (by new-attached we mean just attached to a desktop), ComponentCtrl.redraw(java.io.Writer) [1] is called to render the component, including the widget's class name, all properties, event listeners and so on.

However, you don't have to implement ComponentCtrl.redraw(java.io.Writer) [1] from ground up. AbstractComponent [2] provides a default implementation, so you could override AbstractComponent.renderProperties(org.zkoss.zk.ui.sys.ContentRenderer) [3] instead.

renderProperties

Overriding AbstractComponent.renderProperties(org.zkoss.zk.ui.sys.ContentRenderer) [3] is straightforward: call back super.renderProperties to render inherited properties, and then call one of the render methods to render the properties of the component.

```
protected void renderProperties(ContentRenderer renderer)
throws IOException {
 super.renderProperties(renderer);
 render(renderer, "myProp", _myProp);
 //...
}
```

Notice that the render methods of AbstractComponent [2] will ignore `null`, empty string, and `false` automatically. Thus, the `if` statement in the following example is redundant.

```
if (value != null && value .length() != 0) //redundant since render
will check
    render(renderer, "name", value); //does nothing if null or empty
```

On the other hand, if you want to render null and an empty string, you should invoke the render methods of ContentRenderer [4], such as

```
render.render("name", value);
```

redrawChildren

After calling renderProperties, redraw calls AbstractComponent.redrawChildren(java.io.Writer) [5] to render the properties of children recursively.

Here is the calling sequence of the default implementation of AbstractComponent.redraw(java.io.Writer) [6]:

1. renderProperties(new JsContentRenderer());
2. redrawChildren(out);

Render Special Properties

ZK Client Engine supports several special properties to provide extra functionality, such as late evaluation and so on.

z_al

Specifies a map of properties that should be evaluated after all script files are loaded.

For example,

```
protected void renderProperties(org.zkoss.zk.ui.sys.ContentRenderer
renderer)
throws java.io.IOException {
  //assume js is the JavaScript code snippet
  renderer.renderDirectly("z_al", "{constraint:function(){\nreturn
"+js+";}}");
}
```

Notice that the value of `z_al` is a JavaScript map of properties that will be evaluated, after all the required JavaScript packages are loaded. Moreover, the value of each entry in the map is a function that should return the object being assigned with.

In the above example, the function will be invoked after all packages are loaded, and then the returned value. `js` will be assigned to the `constraint` property.

z_ea

Specifies the property name whose value must be retrieved from the DOM element with the same UUID.

It is typically used to render a property that will be able to be indexed search engines.

For example,

```
renderer.render("z_ea", "content");
```

Then, the value of the `content` property will be retrieved from the inner HTML of the DOM element with the same UUID. Of course, the component has to render the value in the correct DOM element by the use of java.lang.String) Utils.renderCrawlableA(java.lang.String, java.lang.String) [7] or

Utils.renderCrawlableText(java.lang.String) [8].

If the content has to decode first (from < to <), prefix the property name with '$'.

```
renderer.render("z_ea", "$content");
```

z_pk

Specifies a list of packages separated by comma that should be loaded before creating the widgets.

For example,

```
renderer.render("z_pk", "com.foo,com.foo.more");
```

Enforce ZK Update Engine to Redraw a Component

A component can enforce ZK Update Engine to redraw a component by calling Component.invalidate() [9]. Once called, the peer widget will be removed, and a new peer widget will be created to represent the new content. Thus, all modifications to the widget at client will be lost if not preserved (or synchronized back) to the server.

Also notice that Component.redraw(java.io.Writer) [10] won't be called immediately. Rather, ZK Update Engine will accumulate all updates, and then optimize the number of commands (AU responses) that need to be sent.

Dynamic Update a Property

When the application modifies a state that affects the peer widget, a component has to send the updated value to the peer widget. It is done by calling one of the smartUpdate methods of AbstractComponent [2]. For example,

```
public void setValue(String value) {
 if (!_value.equals(value)) {
  _value = value;
  smartUpdate("value", _value);
 }
}
```

If the peer widget was created in the previous request (i.e., the component has been attached to desktop), the invocation of smartUpdate will actually cause the peer widget's setter of the specified properties being called. In the above example, setValue will be called at the client.

On the other hand, if a component is not yet attached to a desktop, smartUpdate will do nothing (since the peer widget doesn't exist). If Component.invalidate() [9] was called, smartUpdate does nothing and previous invocation of smartUpdate of the same request are ignored (since the peer widget will be removed and re-created).

Deferred Property Value

Sometimes the value is not ready when smartUpdate is called, and it is better to retrieve when rendering the components. To defer the evaluation of a value, you can implement DeferredValue [11].

For example, Execution.encodeURL(java.lang.String) [12] is better to be called when rendering components[13] :

```
public void setSrc(String src) {
 if (!Objects.equals(_src, src)) {
  _src = src;
  smartUpdate("src", new EncodedURL());
 }
}
```

```
private class EncodedURL implements DeferedValue {
 public Object getValue() {
  return getDesktop().getExecution().encodeURL(_src);
 }
}
```

[1] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/ComponentCtrl.html#redraw(java.io.Writer)
[2] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/AbstractComponent.html#
[3] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/AbstractComponent.html#renderProperties(org.zkoss.zk.ui.sys.ContentRenderer)
[4] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/ContentRenderer.html#
[5] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/AbstractComponent.html#redrawChildren(java.io.Writer)
[6] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/AbstractComponent.html#redraw(java.io.Writer)
[7] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/impl/Utils.html#renderCrawlableA(java.lang.String,
[8] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/impl/Utils.html#renderCrawlableText(java.lang.String)
[9] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#invalidate()
[10] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#redraw(java.io.Writer)
[11] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/DeferredValue.html#
[12] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#encodeURL(java.lang.String)
[13] It is because smartUpdate is usually called in an event listener, which might run at the event thread (if it is turned on). Meanwhile, WebSphere 5 doesn't allow calling encodeURL other than the servlet thread.

Version History

Version	Date	Content

Client-side

This section describes how to develop the component at the client side (Widget [2]).

See Also

- For introduction, please refer to ZK Component Development Essentials.
- For the server-side component development, please refer to the Server-side section.
- For information about the communication between the client and server, please refer to the Communication section.

Text Styles and Inner Tags

This section is about how to pass the text styles to the inner HTML tags.

Issue

In general, the styles (Widget.setStyle(_global_.String) [1]) are generated directly to the outer DOM element by the use of Widget.domAttrs_(_global_.Map) [2].

However, for some DOM structure, the text-related styles must be specified in some of the inner tags that contain the text. Otherwise, it won't have any effect to the text's styles.

For example, assume that the widget's HTML representation is as follows.

```
<span><input type="checkbox"/><label>Text</label></span>
```

Solution

It can be resolved as follows.

First, generates the style for the inner tag (i.e., <label> in the above case) by calling zk.Widget#domTextStyleAttr_

```
out.push('<label', this.domTextStyleAttr_(), '>',...);
```

Second, override Widget.getTextNode_() [3] to return the DOM element that embeds the text.

```
getTextNode_: function () {
    return zDom.firstChild(this.getNode(), "LABEL");
}
```

Version History

Version	Date	Content

References

[1] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#setStyle(_global_.String)
[2] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#domAttrs_(_global_.Map)
[3] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#getTextNode_()

Rerender Part of Widget

If a widget has a lot of child widgets, the performance will be better if you rerender only the portion(s) that don't have a lot of child widgets (and are not changed).

For example, Groupbox [1] rerenders only itself and the caption child, if any, when `setClosable` is called, as follows.

```
setClosable: function (closable) {
    if (this._closable != closable) {
        this._closable = closable;
        if (this.desktop)
            this.rerender(zk.Skipper.nonCaptionSkipper);
   }
}
```

It invokes Widget.rerender(zk.Skipper) [2] with a skipper (an instance of Skipper [3]). The skipper decides what to skip (i.e., not to rerender), detach the skipped portion(s), and attach them back after rerendering. Thus, the skipped part won't be rerendered, nor unbound/bound, so the performance is better.

In the above example, we use the Skipper.nonCaptionSkipper [4] instance to do the job. It skips all child widgets except the one called caption (i.e., child != this.caption).

In addition to passing a skipper to Widget.rerender(zk.Skipper) [2], the widget has to implement the mold method (redraw) to handle the skipper:

```
function (out, skipper) {
    out.push('<fieldset', this.domAttrs_(), '>');
    var cap = this.caption;
    if (cap) cap.redraw(out);

    out.push('<div id="',  this.uuid, '$cave"', this._contentAttrs(), '>');
    if (!skipper)
        for (var w = this.firstChild; w; w = w.nextSibling)
            if (w != cap) w.redraw(out);
    out.push('</div></fieldset>');
}
```

As shown above, the mold method is also called with the skipper, and the implementation should not redraw the skipped part. In this case, all child widgets except caption are not redrawn.

You can implement your own skipper. Refer to Skipper [3] for details.

Version History

Version	Date	Content

References

[1] http://www.zkoss.org/javadoc/latest/jsdoc/zul/wgt/Groupbox.html#
[2] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#rerender(zk.Skipper)
[3] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Skipper.html#
[4] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Skipper.html#nonCaptionSkipper

Notifications

In this section we discuss the notifications at the client side.

There are three ways to notify: widget events (Event [7]), DOM events (Event [1]) and client activity watches.

A DOM event (Event) is the DOM-level (i.e., low-level) event that is usually triggered by the browser. It is usually listened by the implementation of a widget, rather than the client application.

A widget event is the high-level event. It is used either to encapsulate a DOM event, or to represent a notification specific to a widget, or to an application.

It is generally suggested to listen widget events (rather than DOM events) if possible, since it is easier and more efficient.

A client activity watch is a notification for special activities that are not available as DOM events or widget events, for example, the notification when a widget is becoming invisible.

They are mainly used for component development. Application developers *rarely need* it. For a complete reference, please refer to JavaScript APIs [2].

References

[1] http://www.zkoss.org/javadoc/latest/jsdoc/jq/Event.html#
[2] http://zkoss.org/javadoc/latest/jsdoc/

Widget Events

A widget event is the widget-level event (Event [7]).

Like Event [1] at the server side, the widget event can be anything, and can be triggered by a widget or an application to notify a widget-level or application-level event, such as that a window has been moved.

In addition, ZK Client Engine intercepts most DOM events and encapsulate them into widgets events, such that it is easier and more efficient for component developers to handle events at widget-level (rather than DOM-level, Event [1]).

Event Listening for Component Developers

ZK Client Engine intercepts most DOM events that are targeting the DOM elements belong to widgets. It then encapsulates them into widget events, and then invokes the corresponding method of Widget [2]. For example, when the user moves the mouse over a DOM element of a widget, Widget.doMouseOver_(zk.Event) [2] will be called. Similarly, when the user clicks a DOM element of a widget, Widget.doClick_(zk.Event) [3] will be called.

Listen by Overriding a Method

Thus, the simplest way to listen a DOM event is to override the corresponding method. For example,

```
doMouseDown_: function (evt) {
    //do whatever you want
    this.$supers('doMouseDown_', arguments); //invoke parent.fireX()
and so on
}
```

where evt is an instance of Event [7]. The original DOM event can be retrieved by the use of Event.domEvent [4], and the original DOM element can be found by the use of Event.domTarget [5] (or evt.domEvent.target).

If you want to listen and disable the default behavior, just not to call the super class:

```
doClick_: function (evt) {
    this.fireX(evt);
    //don't call this.$supers to avoid the event propagation
},
```

Note that this approach is suggested for better performance since no real DOM-level event registration is required (as described in the next section).

Event Propgation

The default implementation of the event methods (doXxxx_ in Widget [2]) propagates the event from the target widget to its parent, grandparent and so on. To stop the propagation, you can either invoke Event.stop(_global_.Map) [6], or not calling back the superclass's event method (the effect is the same). In other words, if the propagation is stopped, the parent's event method won't be called.

If a widget event is not stopped and required by the server, it will be sent to the server, and converted to an instance of AuRequest [7] at the server[8] .

In addition to the event propagation, the default implementation will invoke int) Widget.fireX(zk.Event, int) [9] to inoke the application-level listeners, if any (registered with int) Widget.listen(_global_.Map, int) [4].

Notice that there are two kinds of propagation: widget-level and DOM-level. If you stop only the widget-level propagation (by calling evt.stop({propagation:true})), the DOM event will go through all DOM-level event listeners and then trigger the browser default behavior.

---

[1] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/event/Event.html#
[2] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#doMouseOver_(zk.Event)
[3] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#doClick_(zk.Event)
[4] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Event.html#domEvent
[5] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Event.html#domTarget
[6] http://www.zkoss.org/javadoc/latest/zk/zk/Event.html#stop(_global_.Map)
[7] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/au/AuRequest.html#
[8] For more information, please refer to the AU Requests section.
[9] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#fireX(zk.Event,

Capture the Mouse Event

Sometime you want the following Widget.doMouseOver_(zk.Event) (http://www.zkoss.org/javadoc/latest/jsdoc/zk/ Widget. html#doMouseOver_(zk. Event)) and Widget.doMouseUp_(zk.Event) (http:/ / www. zkoss. org/ javadoc/ latest/ jsdoc/ zk/ Widget. html#doMouseUp_(zk. Event)) are called against the same widget, no matter where the mouse-up event happens. This is also known as capturing. It can be done by setting zk.mouseCapture (http://www.zkoss.org/javadoc/latest/jsdoc/_global_/zk.html#mouseCapture) as follows.

```
doMouseDown_: function () {
    zk.mouseCapture = this;
    this.$supers('doMouseDown_', arguments);
}
```

Notice that the mouse capture is reset automatically after Widget.doMouseUp_(zk.Event) (http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#doMouseUp_(zk.Event)) is called.

Capture the Input Event

Sometime you want the following Widget.onKeyPress_(zk.Event) (http://www.zkoss.org/javadoc/latest/jsdoc/zk/ Widget. html#onKeyPress_(zk. Event)) and Widget.onKeyUp_(zk.Event) (http:/ / www. zkoss. org/ javadoc/ latest/ jsdoc/ zk/ Widget. html#onKeyUp_(zk. Event)) to be called against the same widget, no matter where the key-up event happens. It is also known as capturing. It can be done by setting zk.keyCapture (http://www.zkoss.org/javadoc/latest/jsdoc/_global_/zk.html#keyCapture) as follows.

```
doKeyDown_: function () {
    zk.keyCapture = this;
    this.$supers('doKeyDown_', arguments);
}
```

Notice that the key capture is reset automatically after Widget.onKeyUp_(zk.Event) (http:/ / www. zkoss. org/ javadoc/latest/jsdoc/zk/Widget.html#onKeyUp_(zk.Event)) is called.

Events and Corresponding Methods

Events that can be handled by overriding a method

DOM Event Name	Method to Override
blur	Widget.doBlur_(zk.Event) (http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#doBlur_(zk.Event)) Note: unlike others, you have to register a listener with _global_.String, zk.Object) Widget.domListen_(_global_.DOMElement, _global_.String, zk.Object) (http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#domListen_(_global_.DOMElement,) as follows. Otherwise, doBlur_ won't be called. `this.domListen_(n, "onBlur");`
click	Widget.doClick_(zk.Event) (http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#doClick_(zk.Event))
dblclick	Widget.doDoubleClick_(zk.Event) (http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#doDoubleClick_(zk.Event))
contextmenu (aka., the right click)	Widget.doRightClick_(zk.Event) (http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#doRightClick_(zk.Event))
focus	Widget.doFocus_(zk.Event) (http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#doFocus_(zk.Event)) Note: unlike others, you have to register a listener with _global_.String, zk.Object) Widget.domListen_(_global_.DOMElement, _global_.String, zk.Object) (http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#domListen_(_global_.DOMElement,) as follows. Otherwise, doFocus_ won't be called. `this.domListen_(n, "onFocus");`
mouseover	Widget.doMouseOver_(zk.Event) (http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#doMouseOver_(zk.Event))
mouseout	Widget.doMouseOut_(zk.Event) (http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#doMouseOut_(zk.Event))
mousedown	Widget.doMouseDown_(zk.Event) (http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#doMouseDown_(zk.Event))
mouseup	Widget.doMouseUp_(zk.Event) (http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#doMouseUp_(zk.Event))
mousemove	Widget.doMouseMove_(zk.Event) (http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#doMouseMove_(zk.Event))
keydown	Widget.doKeyDown_(zk.Event) (http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#doKeyDown_(zk.Event))
keyup	Widget.doKeyUp_(zk.Event) (http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#doKeyUp_(zk.Event))
keypress	Widget.doKeyPress_(zk.Event) (http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#doKeyPress_(zk.Event))

Event Listening for Application Developers

To listen a widget event, you could invoke int) Widget.listen(_global_.Map, int) (http://www.zkoss.org/javadoc/latest/ jsdoc/ zk/ Widget. html#listen(_global_. Map,) to listen any widget event you want. However, int) Widget.listen(_global_.Map, int) (http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#listen(_global_. Map,) is designed for applications to listen events at the client. Thus, it is also called the application-level event listener.

For component development, the method overriding is suggested as described in the previous subsections.

The signature of an event listener is as follows.

```
function (event) { //an instance of zk.Event
}
```

Event Firing

To fire a widget event, you could invoke zk.Object, _global_.Map, int) Widget.fire(_global_.String, zk.Object, _global_.Map, int) (http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#fire(_global_.String,) or int) Widget.fireX(zk.Event, int) (http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#fireX(zk.Event,).

Then, the listeners registered with int) Widget.listen(_global_.Map, int) (http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#listen(_global_.Map,) will be invoked one-by-one. Then, it will be sent to the server, if an event listener has been registered at the server or it is an import event[1] .

A client-side event listener could stop sending a widget event to the server by invoking Event.stop(_global_.Map) (http://www.zkoss.org/javadoc/latest/jsdoc/zk/Event.html#stop(_global_.Map)) with {au:true}, such as

```
evt.stop({au: true});
```

[1] For more information, please refer to the AU Requests section.

Version History

Version	Date	Content

DOM Events

A DOM event (Event [1]) is the DOM-level event that is usually triggered by the browser. It is usually listened by the implementation of a widget, rather than the client application.

Since ZK Client Engine can intercept most DOM events and encapsulate them into the widget events, it is suggested to listen the widget events, if possible, for better performance (by overriding the corresponding methods, such as Widget.doClick_(zk.Event) [3]). For more information, please refer to the previous section.

How to Listen and Unlisten

There are two different approaches to listen a DOM event: _global_.String, zk.Object) Widget.domListen_(_global_.DOMElement, _global_.String, zk.Object) [1] and jQuery (jq [8]).

Use domListen_ and domUnlisten_

global.String, zk.Object) Widget.domListen_(_global_.DOMElement, _global_.String, zk.Object) [1] registers a DOM-level event listener. The registration should be done when a widget is bound to DOM elements, i.e., when zk.Skipper, _global_.Array) Widget.bind_(zk.Desktop, zk.Skipper, _global_.Array) [2] is called. It is important to un-register by the use of _global_.String, zk.Object) Widget.domUnlisten_(_global_.DOMElement, _global_.String, zk.Object) [3] when a widget is un-bound from DOM elements, i.e., when _global_.Array) Widget.unbind_(zk.Skipper, _global_.Array) [4] is called. For example,

```
bind_: function () {
    this.$supers('bind_', arguments);
    this.domListen_(this.getNode(), "onChange");
},
unbind_: function () {
    this.domUnlisten_(this.node, "onChange");
    this.$supers('unbind_', arguments);
},
_doChange: function (evt) { //event listener
    //evt is an instance of jq.Event
},
```

Unlike jQuery's event listener (jq [8]), _global_.String, zk.Object) Widget.domListen_(_global_.DOMElement, _global_.String, zk.Object) [1] will be ignored if the widget is under control of ZK Weaver (a WYSIWYG editor), i.e., in the so-called *Design Mode*. In most cases, a widget should not register any event listeners when it is under control of ZK Weaver to avoid any conflict.

Use jQuery

The use of jQuery (jq [8]) is similar except using one of the event listening methods found in jQuery [5].

```
bind_: function () {
    this.$supers('bind_', arguments);
    jq(this.$("form")).bind("reset", this.proxy(this._resetForm));
},
unbind_: function () {
    jq(this.$("form")).unbind("reset", this.proxy(this._resetForm));
```

```
    this.$supers('unbind_', arguments);
},
_resetForm: function (evt) { //event listener
   this.doSomething(); //this refers to the widget since this.proxy is
used
},
```

where we use Object.proxy(_global_.Function) [6] to proxy a function such that `this` will refer to the widget when the method is called. Also notice that the event name used with jQuery does not start with `on`.

Version History

Version	Date	Content

References

[1] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#domListen_(_global_.DOMElement,
[2] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#bind_(zk.Desktop,
[3] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#domUnlisten_(_global_.DOMElement,
[4] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#unbind_(zk.Skipper,
[5] http://api.jquery.com/category/events/
[6] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Object.html#proxy(_global_.Function)

Client Activity Watches

In addtion to widget events (Event [7]) and DOM events (Event [1]), there are some special notifications called client activity watches. They are used to notify special activities, such as when a widget becomes invisible, or a window is brought to the top. This kind of activity can be listened by so-called watch (zWatch [1])

Listen and Unlisten

To add a watch (i.e., listen to a client activity), you could use zWatch.listen(_global_.Map) [2] as follows:

```
zWatch.listen({
    onSize: this,
    onShow: this,
    onHide: [this, this._onHide]
});
```

As shown, the key of each entry in the given map is the name of the client activity (aka., the watch name), and the value could be one of the following:

- An object that has a method with the same name. In the above case, **this** must have the `onSize` and `onShow` methods
- A two-element array, where the first element is the target, and the second is the method

The signature of the method is as follows.

```
function onWhatever(ctl, arg0, arg1...) {
  //ctl.origin: the object passed as the first argument to zWatch.fire
or zWatch.fireDown
```

```
  //ctl.fireDown(something) and ctl.fire(something):
  //
}
```

where `ctl` is a controller allowing you to have better control of the invocation sequence of the listeners, and `arg0` and others are the arguments that passed to java.lang.Object, _global_.Map, java.lang.Object...) zWatch.fire(_global_.String, java.lang.Object, _global_.Map, java.lang.Object...) [3] or java.lang.Object, _global_.Map, java.lang.Object...) zWatch.fireDown(_global_.String, java.lang.Object, _global_.Map, java.lang.Object...) [4].

The controller has two methods: fire and fireDown, and one field: origin. The fire and fireDown methods are used to fore the remaining listeners (caused by the same invocation of of java.lang.Object, _global_.Map, java.lang.Object...) zWatch.fire(_global_.String, java.lang.Object, _global_.Map, java.lang.Object...) [3] or java.lang.Object, _global_.Map, java.lang.Object...) zWatch.fireDown(_global_.String, java.lang.Object, _global_.Map, java.lang.Object...) [4]) to be invoked. If your listener doesn't call any of them, the other listeners are called in the same order of registration.

Here is the pseudo code for the controller:

```
interface Controller {
  /** Usually zk.Widget (unless fire and fireDown was called with a
different object) */
  Object origin;
  /** enforce the remaining listeners to be invoked immediately (change
 the invocation sequence) */
  void fire(Object ref, Object...);
  /** enforce the remaining listeners to be invoked immediately (change
 the invocation sequence) */
  void fireDown(Object ref, Object...);
}
```

where `ref` is optional. If specified, it will invoke only the listeners for the given object (and its descendants if fireDown) that are not invoked yet. If null, it will invokes all the listeners that are not invoked yet.

The origin field (`ctl.origin`) is the original object (usually a widget, Widget [2]) passed as the first argument when java.lang.Object, _global_.Map, java.lang.Object...) zWatch.fire(_global_.String, java.lang.Object, _global_.Map, java.lang.Object...) [3] or java.lang.Object, _global_.Map, java.lang.Object...) zWatch.fireDown(_global_.String, java.lang.Object, _global_.Map, java.lang.Object...) [4] was called. In other words, it is the one causes the client activity. It is null if not available.

To unlisten, you could use zWatch.unlisten(_global_.Map) [5] as follows:

```
zWatch.unlisten({
    onSize: this,
    onShow: this,
    onHide: [this, this._onHide]
});
```

Fire

The client activity is triggered (aka., fired) by either java.lang.Object, _global_.Map, java.lang.Object...) zWatch.fire(_global_.String, java.lang.Object, _global_.Map, java.lang.Object...) [3] or java.lang.Object, _global_.Map, java.lang.Object...) zWatch.fireDown(_global_.String, java.lang.Object, _global_.Map, java.lang.Object...) [4].

java.lang.Object, _global_.Map, java.lang.Object...) zWatch.fire(_global_.String, java.lang.Object, _global_.Map, java.lang.Object...) [3] will invoke the listeners for the target object (the first argument), while java.lang.Object, _global_.Map, java.lang.Object...) zWatch.fireDown(_global_.String, java.lang.Object, _global_.Map, java.lang.Object...) [4] will invokes the listeners for the target object and all of its descendants (i.e., the target object's children, grandchildren...).

For example, if a widget resizes itself, it could fire down onSize as follows.

```
zWatch.fireDown("onSize", wgt);
```

The target object could be anything as long as the listener recognizes it, but ZK's standard widgets use Widget [2] only.

Client Activities

Here is the list of client activities that you could watch.

beforeSize

```
[fireDown]
```

It is called right before the browser window or the parent widget is resized.

beforeSize, onFitSize and onSize are fired when the browser window or a widget is resized. beforeSize is fired first, such that the listeners could reset style's width or height. Then, the listeners of onFitSize are called in the reverse order (child first) to caculate the minimal allowed size. Finally, the the listener of onSize can change it to the correct size.

Notice java.lang.Object, _global_.Map, java.lang.Object...) zWatch.fireDown(_global_.String, java.lang.Object, _global_.Map, java.lang.Object...) [4] must be used to fire this event, so only the listeners of descendants of the specified widget will be called.

- Parameters
  - ctl.origin - the widget that causes the resizing. If null, it means the whole browser is resized.

onBindLevelChange

```
[fire]
```

It is called if the bind level of a widget (Widget [2]'s bindLevel) is changed due to moving from one parent to another.

Notice it won't be called if it is unbound and bound (i.e., detached and attached).

Notice java.lang.Object, _global_.Map, java.lang.Object...) zWatch.fire(_global_.String, java.lang.Object, _global_.Map, java.lang.Object...) [3] is used, so all listeners are invoked.

onFitSize

```
[fireDown; reverse order]
[since 5.0.8]
```

It is called between beforeSize and onSize.

beforeSize, onFitSize and onSize are fired when the browser window or a widget is resized. beforeSize is fired first, such that the listeners could reset style's width or height. Then, the listeners of onFitSize are called in the reverse order (child first) to caculate the minimal allowed size. Finally, the the listener of onSize can change it to the correct size.

Notice that the listeners of onFitSize are called in the reverse order, i.e., the child is called before the parent. However, superclass's listener of the same widget will still be called first (like onSize and other events).

- Parameters
  - ctl.origin - the widget that causes the resizing. If null, it means the whole browser is resized.

onHide

```
[fireDown]
```

It is called before a widget is going to become invisible.

Notice java.lang.Object, _global_.Map, java.lang.Object...) zWatch.fireDown(_global_.String, java.lang.Object, _global_.Map, java.lang.Object...) [4] must be used to fire this event, so only the listeners of descendants of wgt will be called.

- Parameters
  - ctl.origin - the widget is becoming invisible
- See Also
  - #onShow

onFloatUp

```
[fire]
```

It is called after a widget has gained the focus. It means the 'float' widget that is the parent of the focus widget shall become topmost.

Notice java.lang.Object, _global_.Map, java.lang.Object...) zWatch.fire(_global_.String, java.lang.Object, _global_.Map, java.lang.Object...) [3] is used, so all listeners are invoked.

- Parameters
  - ctl.origin - the widget gains the focus.

onResponse

```
[fire]
```

It is called after the response of the AU request has been sent back from the server, and processed.

Notice the java.lang.Object, _global_.Map, java.lang.Object...) zWatch.fire(_global_.String, java.lang.Object, _global_.Map, java.lang.Object...) [3] is used, so all listeners are invoked.

onCommandReady

```
[since 7.0.5]
[fire]
```

It is called after the AU commands processed and before "onResponse". In other words, the "onCommandReady" is fired without "setTimeout" which is triggered directly. Unlike "onResponse" will be triggered with a "setTimeout".

Notice the java.lang.Object, _global_.Map, java.lang.Object...) zWatch.fire(_global_.String, java.lang.Object, _global_.Map, java.lang.Object...) [3] is used, so all listeners are invoked.

onRestore

```
[fireDown]
```

It is called when Skipper [3] restores the DOM elements.

It is rarely required but to fix the browser's bug if any. Furthermore, if you listen to onRestore, it is likely you have to listen onVParent too.

- Parameters
  - ctl.origin - the widget has become visible
- See Also
  - #onVParent

onScroll

```
[fire]
```

It is called when the browser window or the specified widget is scrolling.

Notice the java.lang.Object, _global_.Map, java.lang.Object...) zWatch.fire(_global_.String, java.lang.Object, _global_.Map, java.lang.Object...) [3] is used, so all listeners are invoked.

- Parameters
  - ctl.origin - the widget that is scrolling (i.e., causing the onScroll watch), or null if the whole browser window is scrolling

onSend

```
[fire]
```

It is called before sending the AU request to the server. The implicit argument indicates whether all AU requests being sent are implicit.

Notice java.lang.Object, _global_.Map, java.lang.Object...) zWatch.fire(_global_.String, java.lang.Object, _global_.Map, java.lang.Object...) [3] is used, so all listeners are invoked.

onSize

```
[fireDown]
```

It is called when the browser window and a widget is resized.

beforeSize, onFitSize and onSize are fired when the browser window or a widget is resized. beforeSize is fired first, such that the listeners could reset style's width or height. Then, the listeners of onFitSize are called in the reverse order (child first) to caculate the minimal allowed size. Finally, the the listener of onSize can change it to the correct size.

Notice that a layout widget (such as Borderlayout and Hbox) must fire both beforeSize and onSize when it resizes.

Notice java.lang.Object, _global_.Map, java.lang.Object...) zWatch.fireDown(_global_.String, java.lang.Object, _global_.Map, java.lang.Object...) [4] must be used to fire this event, so only the listeners of descendants of wgt will be called.

- Parameters
  - ctl.origin - the widget that causes the resizing. If null, it means the whole browser is resized.

onShow

```
[fireDown]
```

It is called after a widget has become visible.

Notice java.lang.Object, _global_.Map, java.lang.Object...) zWatch.fireDown(_global_.String, java.lang.Object, _global_.Map, java.lang.Object...) [4] must be used to fire this event, so only the listeners of descendants of wgt will be called.

- Parameters
  - ctl.origin - the widget has become visible
- See Also
  - #onHide

onVParent

```
[fireDown]
[since 5.0.8]
```

It is called when jqzk.makeVParent() [6] or jqzk.undoVParent() [7] is called to move a DOM element to/from document.body.

It is rarely required but to fix the browser's bug if any. Furthermore, if you listen to onVParent, it is likely you have to listen onRestore too.

- Parameters
  - ctl.origin - the widget has become visible
- See Also
  - #onRestore

Version History

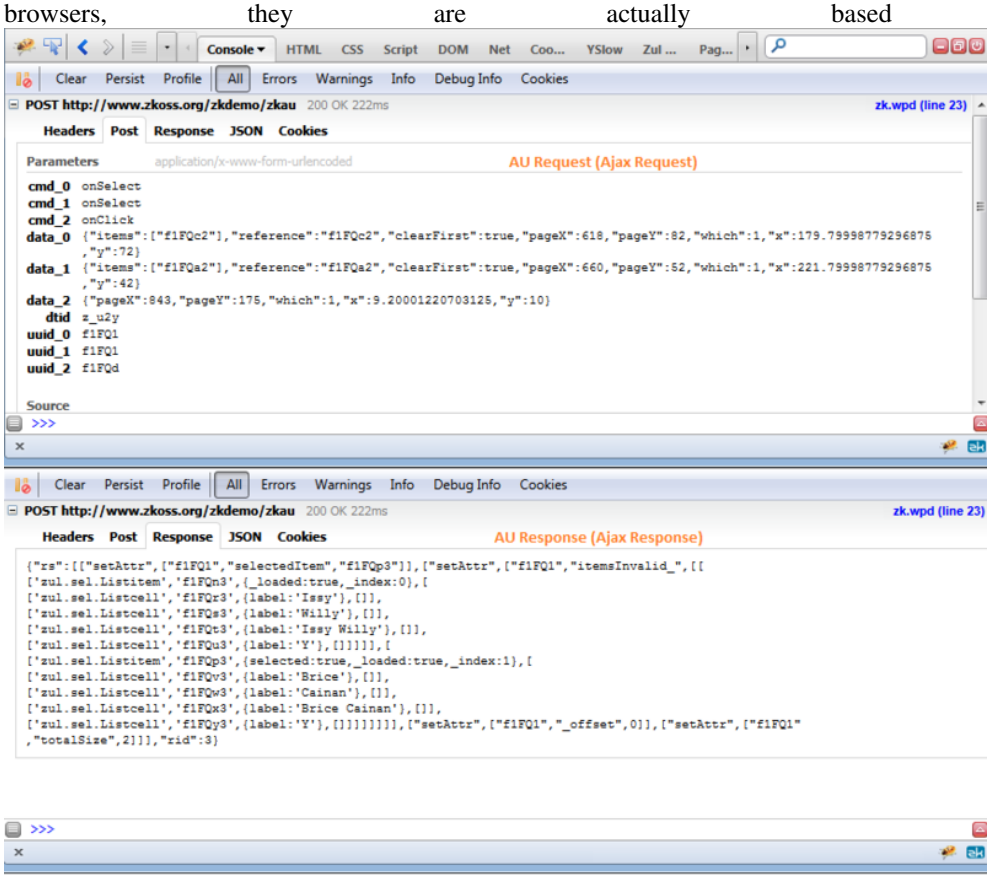
Version	Date	Content
5.0.8	August 2011	onFitSize and onVParent was introduced.
7.0.5	February 2015	Support onCommandReady [8]

References

[1] http://www.zkoss.org/javadoc/latest/jsdoc/_global_/zWatch.html#
[2] http://www.zkoss.org/javadoc/latest/jsdoc/_global_/zWatch.html#listen(_global_.Map)
[3] http://www.zkoss.org/javadoc/latest/jsdoc/_global_/zWatch.html#fire(_global_.String,
[4] http://www.zkoss.org/javadoc/latest/jsdoc/_global_/zWatch.html#fireDown(_global_.String,
[5] http://www.zkoss.org/javadoc/latest/jsdoc/_global_/zWatch.html#unlisten(_global_.Map)
[6] http://www.zkoss.org/javadoc/latest/jsdoc/_global_/jqzk.html#makeVParent()
[7] http://www.zkoss.org/javadoc/latest/jsdoc/_global_/jqzk.html#undoVParent()
[8] http://tracker.zkoss.org/browse/ZK-2516

Communication

This section describes the communication between the server and the clients. The request sent from the client to the server is called the AU requests, while the response from the server to the client is called the AU responses. For browsers, they are actually based on Ajax.



AU Requests

An AU request is a request sent from the client to the server to notify an *event* happening at the client, such as a click, a state change and so on[1] .

[1] For browsers, an AU request is an Ajax request.

Version History

Version	Date	Content

Client-side Firing

In general, an AU request is caused by a widget event (Event [7]) that is going to be sent to the server. This happens when the widget event targets a widget that is created at the server, or with the `toServer` option (specified in Event.opts [1]). In addition, you could invoke zAu.fire() [2] explicitly to fire an AU request to the server.

Fire Event to Widget

An event can be fired to a widget by the use of zk.Object, _global_.Map, int) Widget.fire(_global_.String, zk.Object, _global_.Map, int) [3] and int) Widget.fireX(zk.Event, int) [9]. For example,

```
onCloseClick: function () {
 this.fire('onClose');
}
```

The event will be *propagated* to the widget's parent, parent's parent and so on, until all ancestors are notified, or the propagation has been stopped by Event.stop(_global_.Map) [2].

After the widget and all of its ancestors are notified, this event is converted to an AU request and sent to the server, if

1. The widget has a peer component, i.e., the widget was created by ZK Client Engine because of the instantiation of a component at the server[4] . Notice that, to minimize the traffic, ZK Client Engine sends the AU request only if one of the following conditions is satisfied:
   - The event is declared as an important event (at server).
   - The server has registered an event listener (EventListener [5]) for it.
2. Or, the `toServer` option has been specified in Event.opts [1] of the event. For example,

```
zAu.send(new zk.Event(wgt, "onFoo", {foo: 'my data'}, {toServer:true}));
```

For more information, please refer to the next section.

[1] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Event.html#opts
[2] http://www.zkoss.org/javadoc/latest/zk/_global_/zAu.html#fire()
[3] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#fire(_global_.String,
[4] If a widget is created automatically because of a peer component, Widget.inServer (http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#inServer) will be true.
[5] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/EventListener.html#

Fire Event to Desktop

At the client, a desktop (Desktop (http://www.zkoss.org/javadoc/latest/jsdoc/zk/Desktop.html#)) is also a widget (Widget (http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#)). So, firing an event to a desktop is the same as firing to a widget.

If you would like to fire an event to all desktops, please refer to the next section.

Fire Event Directly to Server

If you would like to fire an event to the server directly, you could invoke int) zAu.send(zk.Event, int) (http://www.zkoss.org/javadoc/latest/jsdoc/_global_/zAu.html#send(zk.Event,). In other words, the event won't go through the target widget's listeners, and will be sent to the server, no matter if it has a peer component or anything else.

The second argument specifies the time to wait before sending the request (unit: milliseconds). If negative, the event won't be sent until anther event with non-negative delay is about to be sent. In other words, if negative, it means the event is deferrable.

If you would like to send an event to all desktops (in the current browser window), you could specify `null` as the target widget of the event.

What States to Send Back the Server

A component has to synchronize every state affecting the widget's behavior to the client, but the widget is required to send to the server only the state that is changed by the user. For better performance and offline capability, it is not necessary to send back the states changed by the client application.

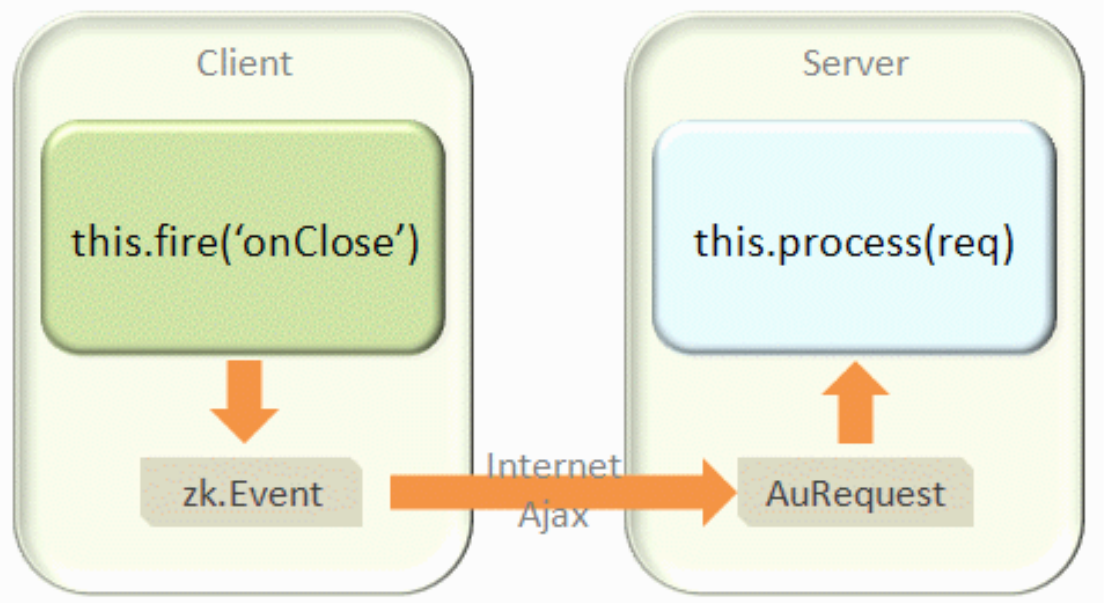
For example, the change of the value of a textbox widget is better to send back to the peer widget since the user might change it. On the other hand, it is not necessary to send the change of the value of a label widget, since the user won't be able to change it.

Version History

Version	Date	Content

Server-side Processing

Process AU Requests at Server



A widget event (Event [7]) is converted to an AU request and then sent to the server. When the event arrives at the server, it is converted to be an instance of AuRequest [7]), and then pass to the desktop for serving by invoking boolean) DesktopCtrl.service(org.zkoss.zk.au.AuRequest, boolean) [1]. If the request is targeting a component, the component's boolean) ComponentCtrl.service(org.zkoss.zk.au.AuRequest, boolean) [2] will then be called to serve it.

Component State Synchronization

Thus, if you implement a component, you could override boolean) ComponentCtrl.service(org.zkoss.zk.au.AuRequest, boolean) [2] to handle it.

Here is an example (from Radio [3]):

```
public void service(org.zkoss.zk.au.AuRequest request, boolean
everError) {
	final String cmd = request.getCommand();
	if (cmd.equals(Events.ON_CHECK)) {
		CheckEvent evt = CheckEvent.getCheckEvent(request);
		_checked = evt.isChecked();
		fixSiblings(_checked, true);
		Events.postEvent(evt);
	} else
		super.service(request, everError);
}
```

Application-level Notification

If the AU request is sent by an application for custom service, you could implement AuService [4] to serve it and then plug it to the targeted component *or* desktop, depending on your requirement. If the request is targeting a desktop, you can only intercept it at the desktop-level. If targeting a component, you could intercept it at either component-level or desktop-level.

Since all requests will be passed through AuService [4] that you plug, the performance of the implementation should be good. In addition, this method should return true if it has been processed to avoid any further processing.

```
public class FooAuService implements AuService {
    public boolean service(AuRequest request, boolean everError) {
        final String cmd = request.getCommand();
        if ("onFoo".equals(cmd)) { //assume onFoo a custom request
            //handle it
            return true; //indicate it has been processed
        }
        return false; //not processed at all
    }
}
```

Intercept at Desktop-level

To plug it to the desktop, you could implement a listener of DesktopInit [5] to add it to a desktop by Desktop.addListener(java.lang.Object) [6]. Then, specify the listener to `WEB-INF/zk.xml`. For example,

```
package foo;
public class FooDesktopInit implements DesktopInit {
    public void init(Desktop desktop,  Object request) throws Exception
 {
        desktop.addListener(new FooAuService()); //assume you have a
custom service called FooAuService
    }
}
```

and, in `WEB-INF/zk.xml`

```
<listener>
      <listener-class>foo.FooDesktopInit</listener-class>
</listener>
```

Intercept at Component-level

To plug it to the component, you could invoke Component.setAuService(org.zkoss.zk.au.AuService) [7].

Client Event Declaration

As described in the previous section, a widget event (Event [7]) will be sent to the server, only if *the server needs it*.

To declare an event that a server *might* need it, you have to invoke java.lang.String, int) AbstractComponent.addClientEvent(java.lang.Class, java.lang.String, int) [8]. It is a static method and usually called in a static clause as shown below.

```
public class A extends LabelImageElement {
      static {
            addClientEvent(A.class, Events.ON_FOCUS, 0);
            addClientEvent(A.class, Events.ON_BLUR, 0);
      }
//...
}
```

Once declared, an event will be sent to the server if one of the following conditions is satisfied:

1. An event listener (EventListener [5]) has been registered at the server.
2. The event has been declared as *important* (see below).

Important Events

Some events that must be sent to the server no matter if an event listener has been registered for it. Typical examples are events that are used to synchronize the states back to the server, such as onChange.

These events are called *important events*. You could declare an event as important by specifying ComponentCtrl.CE_IMPORTANT [9] as follows.

```
static {
      addClientEvent(InputElement.class, Events.ON_CHANGE,
CE_IMPORTANT|CE_REPEAT_IGNORE);
}
```

Notice that the important event won't be sent to the server immediately if it does not have any non-deferrable event listener at the server[10] .

---

[1] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/DesktopCtrl.html#service(org.zkoss.zk.au.AuRequest,
[2] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/ComponentCtrl.html#service(org.zkoss.zk.au.AuRequest,
[3] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Radio.html#
[4] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/au/AuService.html#
[5] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/DesktopInit.html#
[6] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Desktop.html#addListener(java.lang.Object)
[7] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#setAuService(org.zkoss.zk.au.AuService)
[8] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/AbstractComponent.html#addClientEvent(java.lang.Class,
[9] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/ComponentCtrl.html#CE_IMPORTANT
[10] A deferrable event listener is an event listener that also implements Deferrable (http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Deferrable.html#). Please refer to ZK Developer's Reference: Event Listening for details.

Force Event to Send Back

java.lang.String, int) AbstractComponent.addClientEvent(java.lang.Class, java.lang.String, int) (http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/AbstractComponent.html#addClientEvent(java.lang.Class,) is usually used by a component developer since the first argument must be the component's class. For application developers, it is more straightforward by specifying the `toServer` option in Event.opts (http://www.zkoss.org/javadoc/latest/jsdoc/zk/Event.html#opts) when instantiating an event. For example,

```
zAu.send(new zk.Event(wgt, "onFoo", {foo: 'my data'}, {toServer:true}));
```

Version History

Version	Date	Content

JSON

The data of a widget event (Event.data [1]) is serialized to a string (so-called marshal) by JSON [2], when the event is sent back to the server. ZK Update Engine will unmarshal it back to a map. If an entry of the data is an array, it will be converted to a list[3] .

The map of data can be retrieve by the use of AuRequest.getData() [4].

For example, assume we fire an event at the client as follows.

```
wgt.fire('onFly', {x: 10, y: 20});
```

Then, we can retrieve and process it at the server as follows:

```
public class Fly extends AbstractComponet {
  static {
    addClientEvent(Fly.class, "onFly", CE_IMPORTANT); //assume it is an
 important event
  }

  public void service(org.zkoss.zk.au.AuRequest request, boolean
everError) {
    String cmd = request.getCommand();
    if (cmd.equals("onFly")) {
      Map data = request.getData();
      int x = ((Integer)data.get("x")).intValue();
      int y = ((Integer)data.get("y")).intValue();
      //do whatever you want
    } else {
      super.service(request, everError);
    }
  }
}
```

Notice that

- AuRequests [5] provides a collection of utilities to convert it to int, long and boolean.
- An integer number is converted to an instance of Integer if it is not overflow (i.e., less than Integer.MAX_VALUE). Otherwise, Long is assumed.
- A decimal number (with . or e) is converted to an instance of Double.

If the data is not a map, it can be retrieved with the empty key:

Types in JavaScript	Codes in Java
wgt.fire("onFly", "sky");	String sky = (String)request.getData().get("");
wgt.fire("onFly", 123);	Integer val = (Integer)request.getData().get("");
wgt.fire("onFly", ["sky", 123]);	List data = (List)request.getData().get(""); String sky = (String)data.get(0); Integer val = (Integer)data.get(1);
wgt.fire("onFly", {left:'10px', top:20px', more:[1, 2]});	Map data = request.getData(); String left = (String)data.get("left"); String top = (String)data.get("top"); List more = (List)data.get("more"); Integer v1 = (Integer)more.get(0);

```
Map data = request.getData();
String left = (String)data.get("left");
String top = (String)data.get("left");
```

For custom data types, you can implement toJSON (at the client) to convert a JavaScript object to a string in custom way.

```
MyClass.prototype.toJSON = function (key) { //key usually meaningless
   return this.uuid;
};
```

In addition to the default handling, You can add a custom AU request service to a component by calling Component.setAuService(org.zkoss.zk.au.AuService) [7].

---

[1] http://www.zkoss.org/javadoc/latest/jsdoc/zk/Event.html#data
[2] http://www.json.org/js.html
[3] More precisely, they are converted to JSONObject (http://www.zkoss.org/javadoc/latest/zk/org/zkoss/json/JSONObject.html#) (a map) and (a list)
[4] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/au/AuRequest.html#getData()
[5] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/au/AuRequests.html#

Version History

Version	Date	Content

AU Responses

An AU response is the command sent from the server to the client for synchronizing back the server's states and performing some functionality. In response to the AU request sent by the client, the server could send one or multiple AU responses to the client. Each AU response consists of a command and a sequence of data. The command is a string, and the data could be any objects (as long as JSON [1] can handle it).

There are two groups of commands depending on whether the command is applied to a particular widget (Widget [2]), or to the whole browser. For the sake of description, we call the first kind of commands as the widget commands, while the second kind the global commands.

Class	Object	Description
AuCmd0 [2]	zAu.cmd0 [3]	AuCmd0 [2] is the class to handle all global commands (i.e., applied to the whole browser). Furthermore, all global commands are handled by an instance of AuCmd0 [2] called zAu.cmd0 [3].
AuCmd1 [4]	zAu.cmd1 [5]	AuCmd1 [4] is the class to handle all widget commands (i.e., applied to a particular widget). Furthermore, all global commands are handled by an instance of AuCmd1 [4] called zAu.cmd1 [5].

Add a New Command

If you'd like to add a new command, you could simply add a new property to to zAu.cmd0 [3] or zAu.cmd1 [5], depending on your requirement. For example,

```
zk.zAu.cmd0.bookmark = function (bk, replace) {
    //...
};
```

Version History

Version	Date	Content

References

[1] http://www.json.org/
[2] http://www.zkoss.org/javadoc/latest/jsdoc/zk/AuCmd0.html#
[3] http://www.zkoss.org/javadoc/latest/jsdoc/_global_/zAu.html#cmd0
[4] http://www.zkoss.org/javadoc/latest/jsdoc/zk/AuCmd1.html#
[5] http://www.zkoss.org/javadoc/latest/jsdoc/_global_/zAu.html#cmd1

Language Definition

This section describes what a language definition and addon are. It is required for the component component. However, you could skip it if you won't develop components. For more information about component development, please refer to ZK Component Development Essentials.

If you would like to change the default configuration of an ZK application, please refer to ZK Developer's Reference: Packing Code.

A language definition defines a component set (aka., a language). For example, ZUL and XHTML are two component sets.

To define a language definition, you have to prepare a file called /metainfo/zk/lang.xml and makes it available to the classpath (such as in a JAR file, or in WEB-INF/classes of a Web application). In addition, you could specify them in /metainfo/zk/config.xml in the classpath.

A language addon is used to extend a language definition. It should be called /metainfo/zk/lang-addon.xml available to the classpath, specified in WEB-INF/zk.xml (in a Web application), or specified in /metainfo/zk/config.xml (in a JAR file; classpath).

When ZK starts, it will parse all language definitions and then all language addons based on their dependency. A language addon is a variant of a language definition. They are almost the same, except the naming and it must specify the addon name.

Samples

Sample of a Language Definition

Here is a sample (from ZUL's lang.xml):

```
<language>
	<language-name>xul/html</language-name>
	<device-type>ajax</device-type>
	<namespace>http://www.zkoss.org/2005/zul</namespace>
	<extension>zul</extension><!-- the first extension is the major one -->
	<extension>xul</extension>

	<version>
		<version-class>org.zkoss.zul.Version</version-class>
		<version-uid>5.0.6</version-uid>
	</version>

	<javascript package="zk"/>
	<javascript package="zul.lang"/>
	<stylesheet href="~./zul/css/zk.wcs" type="text/css"/>

	<renderer-class>org.zkoss.zul.impl.PageRenderer</renderer-class>

	<label-template>
```

```
            <component-name>label</component-name>
            <component-attribute>value</component-attribute>
      </label-template>
      <macro-template>
            <macro-class>org.zkoss.zk.ui.HtmlMacroComponent</macro-class>
      </macro-template>
      <native-template>
            <native-class>org.zkoss.zk.ui.HtmlNativeComponent</native-class>
      </native-template>

      <component>
            <component-name>a</component-name>
            <component-class>org.zkoss.zul.A</component-class>
            <widget-class>zul.wgt.A</widget-class>
            <text-as>label</text-as><!-- treat text within the element as the label prope
            <mold>
                  <mold-name>default</mold-name>
                  <mold-uri>mold/a.js</mold-uri>
                  <css-uri>css/a.css.dsp</css-uri>
            </mold>
      </component>
</language>
```

Sample of a Language Addon

Here is a sample (from zkmax's lang-addon.xml):

```
<language-addon>
      <addon-name>zkmax</addon-name>
      <depends>zkex</depends>
      <language-name>xul/html</language-name>

      <version>
            <version-class>org.zkoss.zkmax.Version</version-class>
            <version-uid>5.0.5</version-uid>
            <zk-version>5.0.5</zk-version><!-- or later -->
      </version>

      <javascript package="zkmax" merge="true"/>

      <component>
            <component-name>portallayout</component-name>
            <component-class>org.zkoss.zkmax.zul.Portallayout</component-class>
            <widget-class>zkmax.layout.Portallayout</widget-class>
            <mold>
                  <mold-name>default</mold-name>
                  <mold-uri>mold/portallayout.js</mold-uri>
                  <css-uri>css/portallayout.css.dsp</css-uri>
```

```
            </mold>
        </component>
</language-addon>
```

Version History

Version	Date	Content

addon-name

Syntax:

```
<addon-name>a_name</addon-name>
```

```
[Required for a language addon]
```

It specifies the name of a language addon. It is required for a language addon. The name must be unique if it is referenced by other addons (with the depends element).

Version History

Version	Date	Content

component

Syntax:

```
<component>
  <component-name>a_name</component-name>
  <extends>a_name</extends>
  <component-class>a_class_name</component-class>
  <widget-class>a_class_name</widget-class>

  <mold>
    <mold-name>a_mold</mold-name>
    <mold-uri>a_uri</mold-uri>
  </mold>

  <text-as>a_property_name</text-as>

  <property>
    <property-name>a_name</property-name>
    <property-value>a_value</property-value>
  </property>

  <annotation>
    <annotation-name>an_annotation_name</annotation-name>
    <property-name>a_property_name</property-name>
    <attribute>
      <attribute-name>an_annotation_attr_name</attribute-name>
      <attribute-value>an_annotation_attr_value</attribute-value>
    </attribute>
  </annotation>

   <custom-attribute>
    <attribute-name>a_custom_attr_name</attribute-name>
    <attribute-value>a_custom_attr_value</attribute-value>
  </custom-attribute>
</component>
```

It specifies a component definition.

Example,

```
<component>
      <component-name>area</component-name>
      <component-class>org.zkoss.zul.Area</component-class>
      <widget-class>zul.wgt.Area</widget-class>
      <mold>
            <mold-name>default</mold-name>
            <mold-uri>mold/area.js</mold-uri>
      </mold>
```

```
</component>
<component>
	<component-name>bandbox</component-name>
	<extends>bandbox</extends>
	<annotation>
		<annotation-name>default-bind</annotation-name>
		<property-name>value</property-name>
		<attribute>
			<attribute-name>access</attribute-name>
			<attribute-value>both</attribute-value>
		</attribute>
	</annotation>
</component>
```

component-name

```
[Required]
```

The name of the component. It must be unique in the whole language.

extends

```
[Optional]
```

It specifies whether this definition is extending from another definition. If omitted, it is considered a definition of a new component. If specified, it extends from the given component definition (which must be defined first).

Notice that the component's name could be the same as the definition it extends from. If the same, the existent definition is simply overriden (no new component definition is created). It is a useful technique to change a component definition, such as adding annotation, providing the initial properties and so on.

component-class

```
[Required if no extends]
```

It specifies the component's Java class at the server side. It is required if you define a new component.

widget-class

```
[Required if no extends][EL expressions allowed]
```

It specifies the widget's class at the client side. For Ajax clients, it must be a JavaScript class. It is required if you define a new component.

Since EL expressions are allowed, the widget class being associated with a component could be decided at runtime. Please refer to Blog: Totally Different Look per User Without Modifying Application [1] for an example.

property

```
[Optional][EL expressions allowed in the property value]
```

It specifies an initial property. Once the property is specified, the corresponding setter will be called when ZK Loader instantiates from a ZUML document. Of course, if you instantiate it directly in Java, this setting has no effect.

Suppose we want to make all window's border default to `normal`, we could do as follows.

Customization Reference

```
<property>
    <property-name>border</property-name>
    <property-value>normal</property-value>
</property>
```

Another example , to turn off combobox's autocomplete.

```
<component>
    <component-name>combobox</component-name>                    <!-- required -->
    <component-class>org.zkoss.zul.Combobox</component-class>    <!-- required -->
    <widget-class>zul.inp.Combobox</widget-class>                <!-- required -->
    <property>
        <property-name>autocomplete</property-name>
        <property-value>false</property-value>
    </property>
</component>
```

text-as

```
[Optional]
```

It specifies the name of the property to assign the text enclosed by the XML element. If omitted (default), the text will be interpreted as a label and a label component defined in label-template will be used.

For example, if you specify

```
<component>
  <component-name>foo</component-name>
  <text-as>content</text-as>
```

then, the following ZUML document

```
<foo>the content of foo</foo>
```

will cause foo.setContent("the content of foo") to be called (assume foo is an instance of the component).

Version History

Version	Date	Content

References

[1] http://blog.zkoss.org/index.php/2010/08/02/totally-different-look-per-user-without-modifying-application/

depends

Syntax:

```
<depends>a_list_of_addon_names</depends>
```

It specifies what language addons this addon depends on. If specified, this addon will be parsed after all the specified addons are parsed.

Example,

```
<depends>zkex, zkmax</depends>
```

which means this addon won't be parsed until both `zkex` and `zkmax` are parsed.

Version History

Version	Date	Content

device-type

Syntax:

```
<device-type>a_type</device-type>
```

```
 [Required for a language definition]
```

It specifies the device type.

Example,

```
<device-type>ajax</device-type>
```

Version History

Version	Date	Content

extension

Syntax:

```
<extension>a_ext</extension>
```

```
[Required for a language definition]
```

It specifies the extension of a file or URI that should be associated with this language. You could have multiple extensions for a language, and the first one is the default one.

Example,

```
<extension>zul</extension>
<extension>xul</extension>
```

Version History

Version	Date	Content

javascript

Syntax:

```
<javascript package="'''pkg_name'''" [merge="''''false'''|true"]/>
<javascript package="'''pkg_name'''" merge="'''a_package_to_merge_to'''"/>
<javascript package="'''pkg_name'''" [ondemand="''''false'''|true"/>
<javascript src="'''a_uri'''"/>
<javascript>
js_code
</javascript>
```

It specifies the JavaScript package or file that has to be loaded to the client. It could also specify the content (the JavaScript code snippet) directly. Notice that, if specified, it is always loaded, no matter if it is required or not.

Example,

```
<javascript package="zul.box"/>
```

package

```
[Optional]
```

It specifies the package to load.

src

```
[Optional]
```

It specifies the URI of the JavaScript file to load. The URI could contain `~./` (such as "~./foo/whatever.js") to indicate a JavaScript file should be loaded from the classpath.

merge

```
[Optional][Default: false]
```

It is used with the `package` attribute to specify whether the package should be loaded as part of the `zk` package. If not specified, each package will be loaded separatedly. This speeds up the loading if we merge several packages into one.

Since ZK 6, it is allowed to specify the target package in the merge attribute. For example,

```
<javascript package="foo.lang" merge="zul.lang"/>
```

In fact, `merge="true"` is the same as `merge="zk"`, i.e., merged to the `zk` package. Notice that the target package must be a preloaded package. In other words, it must be specified in another `javascript` element (without the `ondemand` attribute). In most cases, you shall use `zk` for packages that can be cached at the client, and use `zul.lang` for packages that shall not be cached, such as your own locale-dependent messages.

For more information, please refer to the Performance Tips section.

ondemand

```
[Optional][Default: false]
```

It is used to 'cancel' the package specified in another `javascript` element. By default, JavaScript packages are loaded on-demand (i.e., when it is required). By specify <javascript;> in a language definition/addon, we could force some packages to load at the beginning. But if you change your mind, you could either remove the `javascript` element, or specify another `javascript` element with ondemand="true".

Version History

Version	Date	Content
6.0.0	September 2011	The merge attribute could be specified with the package's name to merge to, such as `zul.lang`.

javascript-module

Syntax:

```
<javascript-module name="''name''" version="''version''"/>
```

It specifies the version of a JavaScript module. The specified version will be associated with the URL used to load Javascript packages (such as `zul.db.wpd`), such that the browser will reload them if the version is changed.

The name is either a package or the prefix of it. It matches any package that starts with the given name. For example,

```
<javascript-module name="foo" version="1.5.0"/>
```

Then, it matches the packages named `foo`, `foo.one`, `foo.another` or `foo.any.subpkg`.

If you have multiple packages that don't share the same prefix, you could specify multiple `<javascript-module>`.

name

The name of the module. It should be the package name or the prefix of all packages it contains.

version

The version of the module. Notice it cannot contain slash, and it must be legal to be part of URL. It is suggested to limit the use of number, alphabet, dash and dot.

Version History

Version	Date	Content

label-template

Syntax:

```
<label-template>
  <component-name>a_component_name</component-name>
  <component-attribute>a_property</component-attribute>
</label-template>
```

It specifies how to instantiate a label. When the text is found in a ZUML document, ZK Loader will first check if the so-called text-as property is defined. If so, the setter is called to pass the text to the component. If not, this label template is used to instantiate a label for holding the text.

Example,

```
<label-template>
      <component-name>label</component-name>
      <component-attribute>value</component-attribute>
</label-template>
```

component-name

```
[Required]
```

The name of the component definition that represents a label.

component-attribute

```
[Required]
```

The property of the component definition for holding the text.

Version History

Version	Date	Content

language

Syntax:

```
<language>
```

The root element of a language definition.

Version History

Version	Date	Content

language-addon

Syntax:

```
<language-addon>
```

The root element of a language addon.

Version History

Version	Date	Content

language-name

Syntax:

```
<language-name>a_name</language-name>
```

```
[Required]
```

It specifies the name of a language definition. It is required for both a language definition and a language addon. The name must be unique in the whole system.

Example,

```
<language-name>xul/html</language-name>
```

Version History

Version	Date	Content

library-property

Syntax:

```
<library-property>
  <name>a_name</name>
  <value>a_value</value>
</library-property>
```

```
 [Optional]
```

It specifies the library property (java.lang.String) Library.setProperty(java.lang.String, java.lang.String) [1]).

Version History

Version	Date	Content

References

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

macro-template

Syntax:

```
<macro-template>
  <macro-class>a_class_represents_macro</macro-class>
</macro-template>
```

It specifies the class used to instantiate a macro component.

Example,

```
<macro-template>
      <macro-class>org.zkoss.zk.ui.HtmlMacroComponent</macro-class>
</macro-template>
```

macro-class

```
[Required]
```

The class used to instantiate a macro component.

Version History

Version	Date	Content

namespace

Syntax:

```
<namespace>a_namespace</namespace>
```

```
 [Required for a language definition]
```

It specifies the namespace of this language. It is suggested to be an URL. The last part will be considered as a shortcut. Thus, the last part is better to be identifiable

Example, here is the ZUL namespace and it also defines a shortcut.

```
<namespace>http://www.zkoss.org/2005/zul</namespace>
```

Version History

Version	Date	Content

native-template

Syntax:

```
<native-template>
  <native-class>a_class_represents_native</native-class>
</native-template>
```

It specifies the class used to instantiate a native component. The native component is used only when ZK Loader is rendering a ZUML document. After rendering, multiple native components might be merged into one, and it might be replaced by other component to save the memory at the server.

Example,

```
<native-template>
      <native-class>org.zkoss.zk.ui.HtmlNativeComponent</native-class>
</native-template>
```

native-class

```
[Required]
```

The class used to instantiate a natve component.

Version History

Version	Date	Content

renderer-class

Syntax:

```
<renderer-class>a_class'</renderer-class>
```

```
[Required for a language definition]
```

It specifies the Java class used to render a page for the given language. It must implement PageRenderer [1].

Example,

```
<renderer-class>org.zkoss.zul.impl.PageRenderer</renderer-class>
```

Version History

Version	Date	Content

References

[1] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/PageRenderer.html#

stylesheet

Syntax:

```
<stylesheet href="'a_uri'" type="text/css"/>
<stylesheet>
 css_definitions
</stylesheet>
```

It specifies a CSS file that should be loaded to the client, or the CSS content. Notice that, if specified, the CSS file is always loaded.

Example,

```
<stylesheet href="~./zul/css/zk.wcs" type="text/css"/>
```

href

```
[Optional]
```

It specifies the URI of the CSS file to load. The URI could contain `~./` (such as "~./foo/whatever.js") to indicate that a JavaScript file should be loaded from the classpath.

type

```
[Optional]
```

The type of CSS. It is usually `text/css`.

Version History

Version	Date	Content

system-property

Syntax:

```
<system-property>
  <name>a_name</name>
  <value>a_value</value>
</system-property>
```

```
 [Optional]
```

It specifies the system property (java.lang.System).

Version History

Version	Date	Content

version

Syntax:

```
<version>
   <version-class>a_class</version-class>
   <version-uid>a_version</version-uid>
   <zk-version>a_version</zk-version>
</version>
```

```
[Optional]
```

It specifies the version of this language definition or addon. It also controls whether to ignore this document.

First, ZK checks if the specified class (<version-calss>) matches the version (<version-uid>). Second, it checks if ZK's version is the same or larger than the version specified in <zk-version>.

The specified class, if any, must have a static field called `UID`. ZK will compare its value with the version specified in <version-uid>. For example,

```
package foo;
public class MyAddon {
    public static final String UID = "1.0.3";
}
```

Then, you could specify it as follows.

```
<version>
      <version-class>foo.MyAddon</version-class>
```

```
      <version-uid>1.0.3</version-uid>
      <zk-version>5.0.0</zk-version>
</version>
```

which means `foo.MyAddon.UID` must be 1.0.3, and WebApp.getVersion() [1] must be 5.0.0 or later.

Version History

Version	Date	Content

References

[1] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/WebApp.html#getVersion()

zscript

Syntax:

```
<zscript language="Java|Groovy|Python|Ruby|JavaScript">
 the code snippet
</zscript>
```

It specifies the zscript code to be evaluated when the corresponding interpreter being loaded by a page. In other words, it specified the initial zscript that should be evaluated by any other script defined in a ZUML document.

Example,

```
<zscript language="Java">
import java.util.*;
import java.lang.*;
import org.zkoss.zk.ui.util.Clients;
import org.zkoss.zk.ui.event.*;
import org.zkoss.zk.ui.*;
import org.zkoss.zul.*;

void alert(Object m) {
      Messagebox.show("" + m);
}
</zscript>
```

Version History

Version	Date	Content

Widget Package Descriptor

This section describes what a Widget Package Descriptor is. This is required for the component. However, you could skip it if you do not have to develop components. For more information about component development, please refer to ZK Component Development Essentials.

The Widget Package Descriptor (WPD) is a file describing the information of a package, such as its widget classes and external JavaScript files. WPD must be named **zk.wpd** and placed in the same directory as the widget classes. For example we would place it under **web/js/com/foo**.

Below is an example **zk.wpd** of our SimpleLabel.

```
<package name="com.foo" language="xul/html">
	<widget name="SimpleLabel"/>
</package>
```

The table below describes the elements used within the above XML and their descriptions.

Name	Description
package	The root element denotes the package name and the language it belongs to
widget	The widget class name (without the package name). If the package contains multiple widgets list them one by one

Having created the configuration the basic implementation of our component is complete. However it doesn't have any interactive events. Therefore the next logical step is to start adding events to the component.

Package Dependence

It is common for JavaScript packages to depend on another package. For example, `zul.grid` depends on `zul.mesh` and `zul.menu`. This can easily be specified by placing them within the `depends` attribute as follows.

```
<package name="zul.grid" language="xul/html" depends="zul.mesh,zul.menu">
	<widget name="Column"/>
	<widget name="Columns"/>
	<widget name="Grid"/>
	<widget name="Row"/>
	<widget name="Rows"/>
	<widget name="Foot"/>
	<widget name="Footer"/>
</package>
```

Including additional JavaScript files

If a JavaScript package has to include other JavaScript files, this can be done easily by specifying the file with the `script` element. For example, the following is the content of `zul.db`'s WPD:

```
<package name="zul.db" language="xul/html" depends="zk.fmt,zul.inp">
	<script src="datefmt.js"/>
	<widget name="Calendar"/>
	<widget name="Datebox"/>
```

```
</package>
```

function

syntax

```
<function class="''foo.MyClass''" singature="''java.lang.String'' ''funcName''(''Class0'', ''Class1'')"/>
```

Specifies a static method (aka., a function) that should be called when a WPD file is interpreted. The returned string will be generated directly to the output. In other words, it must be a valid JavaScript code snippet.

Example,

```
<package name="zul.lang" cacheable="false">
      <script src="msgzul*.js"/>
      <function class="org.zkoss.zul.impl.Utils"
            signature="java.lang.String outLocaleJavaScript()"/>
</package>
```

class

```
[Required]
```

The name of the class where the static method is declared.

signature

```
[Required]
```

The signature of the static method. The return type has to be a string and the return value should be a valid JavaScript code snippet.

The method might have arbitrary numbers of arguments. WPD will check the type of each argument and assign a proper value if possible. The following is the type WPD recognized:

Argument Type	Value
javax.servlet.ServletRequest and derives	The current request.
javax.servlet.ServletResponse and derives	The current response.
javax.servlet.ServletContext	The current servlet context.
Others	null

Version History

Version	Date	Content

package

Syntax:

```
<package name="''a_name''" [language="''a_lang''"] [depends="''pkg0'', ''pkg1''..."] [cacheable="''true''|false"]>
```

The root element of a WPD document. It specifies the name of the package, what packages it depends and other information.

Example,

```
<package name="zul.box" language="xul/html" depends="zul,zul.wgt">
      <script src="Layout.js" jsdoc="true"/>

      <widget name="Box"/>
      <widget name="Splitter"/>
      <widget name="Hlayout"/>
      <widget name="Vlayout"/>
</package>
```

name

```
[Required]
```

The package name, such as `zul.grid`. It has to be unique.

language

```
[Optional]
```

The language name, such as xul/html.

If omitted, it means it does not belong to a particular language. It is better to specify one if the WPD document defines a widget.

depends

```
[Optional]
```

It specifies a list of packages that this package depends on. In other words, the packages specified in this attribute will be loaded before loading this package.

cacheable

```
[Optiona][Default: true]
```

It specifies whether the client is allowed to cache the output of this WPD file. By default, it is cacheable since the performance is better. However, you have to turn it off, if you use a function that will generate the output depending on a varying condition (such as Locale and time zone).

Version History

Version	Date	Content

script

Syntax:

```
<script [src="''foo.js''"]
 [browser="ie|ie6|ie6-|ie7|ie7-|ie8|ie8-|gecko|gecko2|gecko2-|gecko3|gecko3-|gecko3.5|opera|safari"]>
```

The script element is used to specify an external JavaScript file, or to embed the JavaScript code directly.

Example,

```
  <script src="a.js"/>
  <script>
function doIt() {
}
  </script>
```

src

```
[Optional]
```

The path of an external JavaScript file. It is suggested to use a path related to the directory of the WPD file. For example,

```
 <script src="abc.js"/>
 <script src="../foo/def.js"/>
```

browser

```
[Optional]
```

It specifies the condition to embed the specified JavaScript file. For example, if browser="ie6-" is specified, the JavaSript file is embeded only if the client is Internet Explorer 6. For the available types you could check, please refer to java.lang.String) Servlets.isBrowser(javax.servlet.ServletRequest, java.lang.String) [1].

Version History

Version	Date	Content

References

[1] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/web/servlet/Servlets.html#isBrowser(javax.servlet.ServletRequest,

widget

Syntax:

```
<widget name="''widgetName''"/>
```

It specifies the widget's name.

Example,

```
<package name="zul.wgt" language="xul/html" depends="zul">
      <widget name="A"/>
      <widget name="Cell"/>
      <widget name="Div"/>
      <widget name="Span"/>
</package>
```

A widget declaration will cause WPD to generate the widget definition in JavaScript. It also assumes that there is a JavaScript file with the same name in the same directory. For example, the above example will cause WPD to embed A.js, Cell.js, Div.js and Span.js.

Version History

Version	Date	Content

Article Sources and Contributors

ZK Client-side Reference *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference *Contributors*: Alicelin, Southerncrossie, Sphota, Tmillsclare, Tomyeh

Introduction *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Introduction *Contributors*: Alicelin, Tomyeh

New to JavaScript *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Introduction/New_to_JavaScript *Contributors*: Alicelin, Sphota, Tomyeh

Object Oriented Programming in JavaScript *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Introduction/Object_Oriented_Programming_in_JavaScript *Contributors*: Alicelin, Tomyeh

Debugging *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Introduction/Debugging *Contributors*: Alicelin, Jumperchen, Tomyeh, Tonyq

General Control *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/General_Control *Contributors*: Alicelin, Tomyeh

UI Composing *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/General_Control/UI_Composing *Contributors*: Alicelin, Ashishd, Tomyeh

Event Listening *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/General_Control/Event_Listening *Contributors*: Alicelin, Tomyeh

Widget Customization *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/General_Control/Widget_Customization *Contributors*: Alicelin, Flyworld, Robertwenzel, Tmillsclare, Tomyeh

JavaScript Packaging *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/General_Control/JavaScript_Packaging *Contributors*: Alicelin, Tomyeh

iZUML *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/General_Control/iZUML *Contributors*: Alicelin, Char, Tomyeh

Customization *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Customization *Contributors*: Tomyeh

Actions and Effects *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Customization/Actions_and_Effects *Contributors*: Alicelin, Tomyeh

Alphafix for IE6 *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Customization/Alphafix_for_IE6 *Contributors*: Alicelin, Tomyeh

Drag-and-Drop Effects *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Customization/Drag-and-Drop_Effects *Contributors*: Alicelin, Tomyeh

Stackup and Shadow *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Customization/Stackup_and_Shadow *Contributors*: Alicelin, Tomyeh

Component Development *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Component_Development *Contributors*: Tomyeh

Components and Widgets *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Component_Development/Components_and_Widgets *Contributors*: Alicelin, Tmillsclare, Tomyeh

Server-side *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Component_Development/Server-side *Contributors*: Tomyeh

Property Rendering *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Component_Development/Server-side/Property_Rendering *Contributors*: Alicelin, Tomyeh

Client-side *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Component_Development/Client-side *Contributors*: Tomyeh

Text Styles and Inner Tags *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Component_Development/Client-side/Text_Styles_and_Inner_Tags *Contributors*: Alicelin, Tomyeh

Rerender Part of Widget *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Component_Development/Client-side/Rerender_Part_of_Widget *Contributors*: Alicelin, Tomyeh

Notifications *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Notifications *Contributors*: Alicelin, Tomyeh

Widget Events *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Notifications/Widget_Events *Contributors*: Alicelin, Tomyeh

DOM Events *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Notifications/DOM_Events *Contributors*: Alicelin, Tomyeh

Client Activity Watches *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Notifications/Client_Activity_Watches *Contributors*: Alicelin, Hawk, Jumperchen, Robertwenzel, Tomyeh

Communication *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Communication *Contributors*: Alicelin, Flyworld, Tomyeh

AU Requests *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Communication/AU_Requests *Contributors*: Flyworld, Tomyeh

Client-side Firing *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Communication/AU_Requests/Client-side_Firing *Contributors*: Alicelin, Tomyeh, Vincent

Server-side Processing *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Communication/AU_Requests/Server-side_Processing *Contributors*: Alicelin, Tomyeh

JSON *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Communication/AU_Requests/JSON *Contributors*: Alicelin, Tomyeh

AU Responses *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Communication/AU_Responses *Contributors*: Alicelin, Tomyeh

Language Definition *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Language_Definition *Contributors*: Alicelin, SimonPai, Tomyeh

Samples *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Language_Definition/Samples *Contributors*: Tomyeh

addon-name *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Language_Definition/addon-name *Contributors*: Tomyeh

component *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Language_Definition/component *Contributors*: Alicelin, Tomyeh, Tonyq

depends *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Language_Definition/depends *Contributors*: Alicelin, Tomyeh

device-type *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Language_Definition/device-type *Contributors*: Tomyeh

extension *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Language_Definition/extension *Contributors*: Alicelin, Tomyeh

javascript *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Language_Definition/javascript *Contributors*: Alicelin, Tomyeh

javascript-module *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Language_Definition/javascript-module *Contributors*: Alicelin, Tomyeh

label-template *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Language_Definition/label-template *Contributors*: Alicelin, Tomyeh

language *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Language_Definition/language *Contributors*: Tomyeh

language-addon *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Language_Definition/language-addon *Contributors*: Tomyeh

language-name *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Language_Definition/language-name *Contributors*: Hawk, Tomyeh

library-property *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Language_Definition/library-property *Contributors*: Tomyeh

macro-template *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Language_Definition/macro-template *Contributors*: Tomyeh

namespace *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Language_Definition/namespace *Contributors*: Tomyeh

native-template *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Language_Definition/native-template *Contributors*: Alicelin, Tomyeh

renderer-class *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Language_Definition/renderer-class *Contributors*: Tomyeh

stylesheet *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Language_Definition/stylesheet *Contributors*: Alicelin, Tomyeh

system-property *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Language_Definition/system-property *Contributors*: Tomyeh

version *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Language_Definition/version *Contributors*: Dennischen, MontyPan, Tomyeh

zscript *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Language_Definition/zscript *Contributors*: Alicelin, Tomyeh

Widget Package Descriptor *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Widget_Package_Descriptor *Contributors*: Alicelin, Peterkuo, Tomyeh

function *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Widget_Package_Descriptor/function *Contributors*: Alicelin, Tomyeh

package *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Widget_Package_Descriptor/package *Contributors*: Alicelin, Tomyeh

script *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Widget_Package_Descriptor/script *Contributors*: Tomyeh

widget *Source*: http://new.zkoss.org/index.php?title=ZK_Client-side_Reference/Widget_Package_Descriptor/widget *Contributors*: Alicelin, Tomyeh

Image Sources, Licenses and Contributors

File:ZKComDevEss_widget_component_application.png *Source*: http://new.zkoss.org/index.php?title=File:ZKComDevEss_widget_component_application.png *License*: unknown *Contributors*: Tmillsclare

Image:UseStack-obscue-1.jpg *Source*: http://new.zkoss.org/index.php?title=File:UseStack-obscue-1.jpg *License*: unknown *Contributors*: Tomyeh

Image:UseStack-ok-1.jpg *Source*: http://new.zkoss.org/index.php?title=File:UseStack-ok-1.jpg *License*: unknown *Contributors*: Tomyeh

Image:UseStack-autohide-1.jpg *Source*: http://new.zkoss.org/index.php?title=File:UseStack-autohide-1.jpg *License*: unknown *Contributors*: Tomyeh

Image: ChangeLabelFlow.png *Source*: http://new.zkoss.org/index.php?title=File:ChangeLabelFlow.png *License*: unknown *Contributors*: Tmillsclare

Image: WidgetAndComponent2.png *Source*: http://new.zkoss.org/index.php?title=File:WidgetAndComponent2.png *License*: unknown *Contributors*: Tmillsclare

Image: WidgetWithoutComponent.png *Source*: http://new.zkoss.org/index.php?title=File:WidgetWithoutComponent.png *License*: unknown *Contributors*: Tmillsclare

Image: WidgetComponentDOM.png *Source*: http://new.zkoss.org/index.php?title=File:WidgetComponentDOM.png *License*: unknown *Contributors*: Tmillsclare

File:Communication.png *Source*: http://new.zkoss.org/index.php?title=File:Communication.png *License*: unknown *Contributors*: Flyworld

Image:ClientEventAuRequest.png *Source*: http://new.zkoss.org/index.php?title=File:ClientEventAuRequest.png *License*: unknown *Contributors*: Tomyeh