

ZK Developer's Reference

For ZK 10.0.0

Contents

Articles

ZK Developer's Reference	1
Overture	1
Architecture Overview	2
Technology Guidelines	6
Extensions	11
Example Project	14
UI Composing	14
Component-based UI	14
ID Space	20
ZUML	24
XML Background	25
Basic Rules	28
EL Expressions	31
Scripts in ZUML	36
Conditional Evaluation	41
Iterative Evaluation	43
On-demand Evaluation	46
Include a Page	49
Load ZUML in Java	51
XML Namespaces	54
Richlet	56
Macro Component	60
Inline Macros	64
Implement Custom Java Class	66
Composite Component	70
Client-side UI Composing	75
Shadow for MVC	75
Event Handling	80
Event Listening	81
Event Firing	85
Event Forwarding	87
Event Queues	89
Client-side Event Listening	97
MVC	97

Controller	99
Composer	99
Wire Components	107
Wire Variables	114
Wire Event Listeners	118
Subscribe to EventQueues	120
Model	123
List Model	126
Groups Model	131
Tree Model	141
Chart Model	151
Matrix Model	152
View	155
Template	156
Listbox Template	157
Grid Template	162
Tree Template	163
Combobox Template	164
Selectbox Template	165
Biglistbox Template	165
Chosenbox Template	166
Tabbox Template	167
Organigram Template	169
Searchbox Template	170
Renderer	171
Listbox Renderer	171
Grid Renderer	172
Tree Renderer	173
Combobox Renderer	174
Tabbox Renderer	175
Organigram Renderer	176
Biglistbox Renderer	177
Item Renderer	178
Stateless Components	179
Building Stateless UI	184
Annotations	188
Annotate in ZUML	188
Annotate in Java	190

Retrieve Annotations	191
Annotate Component Definitions	191
UI Patterns	192
Responsive Design	192
Message Box	202
Layouts and Containers	203
Hflex and Vflex	216
Grid's Columns and Hflex	226
Tooltips, Context Menus and Popups	233
Keystroke Handling	239
Drag and Drop	243
Page Initialization	246
Forward and Redirect	248
File Upload and Download	251
Browser Information and Control	254
Browser History Management	258
Session Timeout Management	261
Error Handling	264
Actions and Effects	269
Useful Java Utilities	272
HTML Tags	278
The html Component	279
The native Namespace	281
The XHTML Component Set	284
Long Operations	286
Use Echo Events	287
Use Event Queues	288
Use Piggyback	291
Communication	292
Inter-Page Communication	292
Inter-Desktop Communication	294
Inter-Application Communication	296
Templating	297
Composition	298
Templates	300
XML Output	302
Event Threads	304
Modal Windows	305

Message Box	306
File Upload	307
Theming and Styling	309
Molds	309
CSS Classes and Styles	310
ZK Official Themes	312
Switching Themes	322
Customizing Standard Themes	324
Creating Custom Themes	327
Theme Template	327
Archive-based Themes	328
Folder-based Themes	332
Understanding the Theming Subsystem	336
Information about a Theme	337
Registering your Theme	338
Providing Theme Resources	339
Resolving Theme URLs	345
Internationalization	346
Locale	346
Time Zone	349
Handling Server and User Time Zones	351
Labels	354
The Format of Properties Files	361
Date and Time Formatting	363
The First Day of the Week	366
Locale-Dependent Resources	368
Warning and Error Messages	370
Server Push	372
Event Queues	373
Synchronous Tasks	374
Asynchronous Tasks	376
Configuration	377
Clustering	381
ZK Configuration	382
Server Configuration	384
Programming Tips	385
Integration	389
Accessing Java EE Scope Objects	389

Presentation Layer	390
Bootstrap	390
JSP	391
Struts	396
Portal	398
ZK Filter	400
Foreign Templating Framework	401
Middleware Layer	406
Spring	406
CDI	411
EJB	415
Persistence Layer	417
JDBC	417
Hibernate	424
JPA	437
Security	446
Spring Security	446
Miscellaneous	458
Google Analytics	458
Start Execution in Foreign Ajax Channel	459
Websocket Channel	461
Performance Tips	467
Use Compiled Java Codes	467
Use Native Namespace instead of XHTML Namespace	469
Use ZK JSP Tags instead of ZK Filter	471
Defer the Creation of Child Components	473
Defer the Rendering of Client Widgets	473
Client Render on Demand	475
Listbox, Grid and Tree for Huge Data	477
Use Live Data and Paging	477
Turn on Render on Demand	477
Implement ListModel and TreeModel	481
Minimize Number of JavaScript Files to Load	485
Load JavaScript and CSS from Server Nearby	487
Specify Stubonly for Client-only Components	489
Reuse Desktops	492
Control resource caching	493
Miscellaneous	495

Security Tips	496
Cross-site scripting	496
Block Request for Inaccessible Widgets	498
Denial Of Service	500
Cross-site Request Forgery	502
OWASP Top 10 Security Concerns In 2017	503
Content Security Policy	505
SSO Redirect Handling	507
Performance Monitoring	509
Performance Meters	510
Event Interceptors	512
Loading Monitors	513
Step by Step Trouble Shooting	513
Accessibility	529
Keyboard Support	532
High Contrast Theme	533
Testing	534
Testing Tips	534
ZATS	537
Upgrade Tips	540
Version Upgrade	540
Edition Upgrade	543
Customization	545
Packing Code	545
Component Properties	546
UI Factory	549
Life Cycle Listener	550
AU Services	552
AU Extensions	553
How to Build ZK Source Code	553
Handle AU Request Resend	554
Supporting Utilities	555
Logger	555
DSP	559
iDOM	561
Common Error Messages	561
Development-time Best Practices	564
Packaging Applications	566

References

Article Sources and Contributors	575
Image Sources, Licenses and Contributors	581

ZK Developer's Reference

If you are new to ZK, you might want to take a look at the Getting Started ^[1] first.

Documentation:Books/ZK_Developer's_Reference

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

<comment>http://books.zkoss.org/wiki/ZK_Developer's_Reference</comment>

References

[1] https://www.zkoss.org/documentation#Getting_Started

Overture

The ZK Developer's Reference is a reference of general features and advanced topics.

If you are new to ZK, you might want to start with the Tutorial ^[1] and ZK Essentials ^[2] first.

For the information of individual component, please refer to ZK Component Reference.

For information on ZUML, please refer to the ZUML Reference.

References

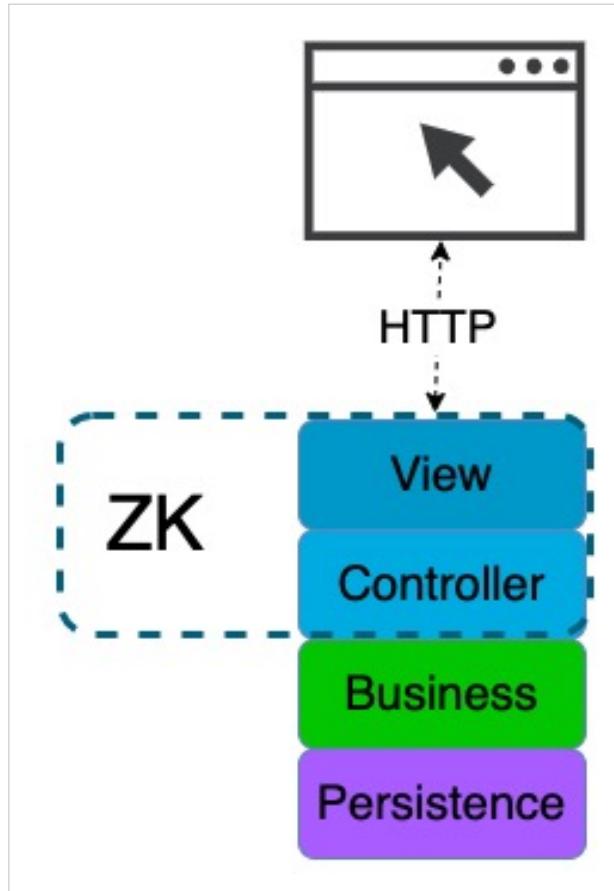
[1] http://www.zkoss.org/documentation#Getting_Started

[2] <http://books.zkoss.org/zkessentials-book/master/>

Architecture Overview

From the Application Developer's Perspective

Under the multi-tier Java EE web application architecture, ZK framework belongs to web tier:



Because ZK is designed to have a clear separation of responsibility and provides controller/ViewModel to integrate the business and persistence tier, you can use any business and persistence layer framework, Java library, or database.

The ZK application runs on the server. It can access backend resources, assemble UI with components, listen to user's activities, and then manipulate components to update UI. All of the above activities can be accomplished on the server. The synchronization of components' states between the browser and the server is done automatically by ZK and transparently to the application.

When running on the server, the application has access to full Java technology stacks. User activities, such as Ajax and Server Push, are abstracted to event objects. UI is composed by POJO-like components. It is the most productive approach to develop a modern Web application.

With ZK's **Server+client Fusion architecture**, your application will never stop running on the server. The application can enhance interactivity by adding optional client-side functionality, such as client-side event handling, visual effect customizing or even UI composing without server-side coding. ZK enables seamless fusions ranging from pure server-centric to pure client-centric. You can have the best of two worlds: productivity and flexibility.

From the Component Developer's Perspective

Each UI object in ZK consists of a component and a widget. A component is a Java object running on the server representing a UI object which can be manipulated by a Java application. A component has all the behavior of a UI object except that it does not have a visual part. A widget is a JavaScript object^[1] running at the client. This object represents the UI object which interacts with the user. Therefore, a widget usually has a visual appearance and it handles events happening at the client.

The relationship between a component and a widget is one-to-one. However, if a component is not attached to a page, there will not be a corresponding widget at the client. On the other hand, the application is allowed to instantiate widgets at the client directly without a corresponding component.

How state synchronization and load distribution might occur depends really on the component. The ZK Client Engine and the Update Engine will work together to provide an elegant and robust channel to simplify the implementation.

For example, assuming that we want to implement a component that allows a user to click on a DOM element to show some detailed information of a component, there are at least two approaches to implement it. Firstly, we could load the detailed information to the client when the widget is instantiated, and then show the details with pure client code. Alternatively, we may choose not to load the detailed information at the very beginning before sending a request back to the server for fulfilling the details when the user clicks on it.

Obviously, the first approach consumes more bandwidth at the initial request but at the same time it also provides faster responses when users click on the DOM element. This is generally more transparent to the application developers, and the implementation of a component can be enhanced later as the project progresses.

[1] It depends on the client. For Ajax-enabled browsers, it is a JavaScript object. For ZK Reach for Android (<http://code.google.com/p/zkreach/>), it is a Java object running on an Android device.

Execution Flow of Loading a Page

1. When a user visits a zk page (zul/zhtml) in a browser, a request is sent to the Web server. If the requested URL matches with the ZK's configured URL pattern. A ZK loader will be invoked to serve this request.

For more information, please refer to ZK Configuration Reference

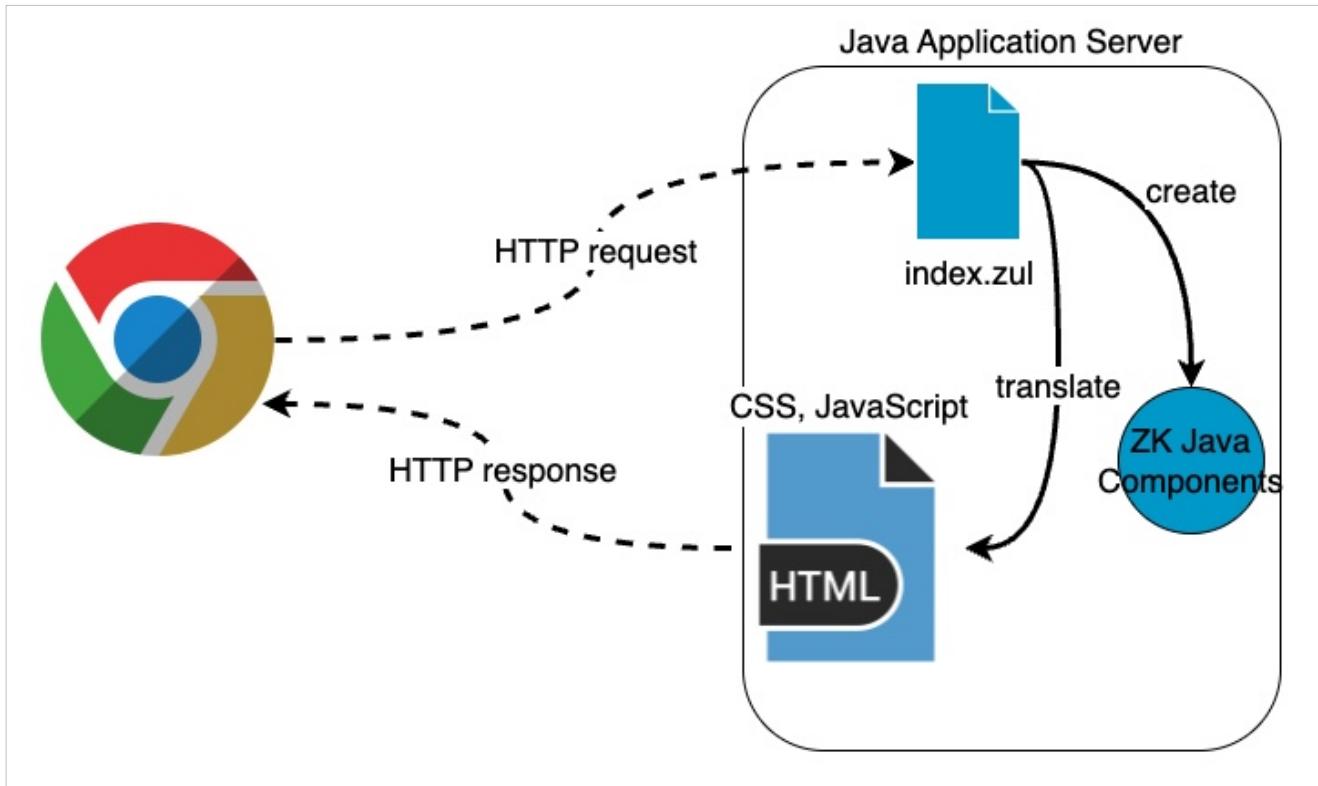
2. The ZK loader loads a specified page and interprets that page to create ZK components accordingly and instantiates specified controllers (Composer/ViewModel).

If a URL is mapped to a richlet, ZK invokes the richlet to handle all UI composition. For more information, please refer to Richlet.

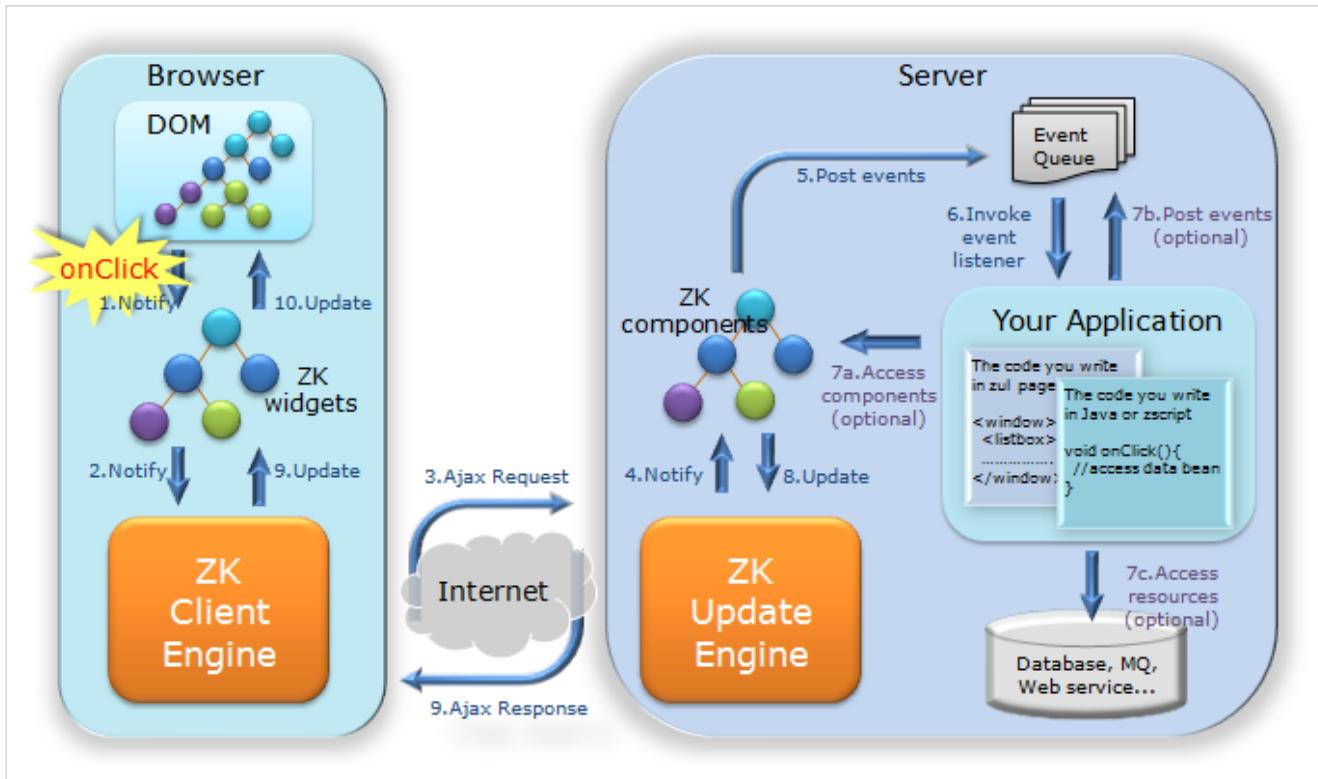
3. After interpreting the whole page, the ZK loader will render the result to an HTML page. The HTML page is then sent back to the browser accompanied by the ZK Client Engine.

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

4. The ZK Client Engine renders the widgets into DOM elements and then inserts the DOM elements into the browser's DOM tree to make them visible to users.
5. After that, the ZK Client Engine will sit at the browser to serve requests made by the users, widgets, or applications. If it goes to another page, the execution flow will start over again. If it is going to send an Ajax request back, another execution flow will start as described in the following section.



Execution Flow of Serving an Ajax Request



1. The execution flow starts from a widget or the application. This is usually caused by the user's activity (or the application's requirement) and is done by posting a client-side event (Event (<http://www.zkoss.org/javadoc/latest/javadoc/zk/Event.html#>)) to a widget (Widget.fire(zk.Event,int) (<http://www.zkoss.org/javadoc/latest/>))

- jsdoc/zk/Widget.html#fire(zk.Event,int))).
2. The event is then bubbled up to the widget's parent, parent's parent, and finally the ZK Client Engine^[1]. The ZK Client Engine then decides whether and when to send the event back to the server in an Ajax request.
 3. If necessary, the ZK Client Engine will send an Ajax request to the ZK Update Engine on the server^[2].
 4. Upon receiving Ajax requests, the ZK Update engine will invoke ComponentCtrl.service(org.zkoss.zk.au.AuRequest,boolean) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/ComponentCtrl.html#service\(org.zkoss.zk.au.AuRequest,boolean\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/ComponentCtrl.html#service(org.zkoss.zk.au.AuRequest,boolean))) for handling an AU request. ZK also wraps a AU request into Execution (<https://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html>) object.
 5. How the AU request can be handled is really up to a component. But, the component that handles the request usually updates the states, if necessary, and then notifies the application by posting events to the current execution (Events.postEvent(org.zkoss.zk.ui.event.Event) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Events.html#postEvent\(org.zkoss.zk.ui.event.Event\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Events.html#postEvent(org.zkoss.zk.ui.event.Event)))).
 6. If any event is posted, the ZK Update Engine will process them one-by-one by invoking the event listeners.
 7. The event listener, provided by an application, may choose either to update the backend resources or the components or to post other events.
 8. Finally, the ZK Update Engine collects all updates of components, including states change, attachment and detachment for optimization and then send a collection of commands back to the client.
 9. The ZK Client Engine evaluates each of these commands to update the widgets accordingly. Then the widgets will update the browser's DOM tree to make them available to the user.

[1] A widget could choose to stop this bubble-up propagation by use of UNIQ-javadoc-0-736dd3f9533a2ca2-QINU

[2] . From the server's viewpoint, an Ajax request is another type of HTTP request.

When to Send an Ajax Request

When the ZK Client Engine receives a bubbled-up client-side event (Event (<http://www.zkoss.org/javadoc/latest/jsdoc/zk/Event.html#>)), it will decide whether and when to send the event back to the server for further processing:

1. If there is a non-deferrable event listener registered on the server, the Ajax request will be sent immediately.
2. If there is a deferrable event listener registered on the server, the request will be queued at the client and it will be sent when another event is triggered and a non-deferrable event listener registered for it.
3. If the widget declares that the event is important (ComponentCtrl.CE_IMPORTANT (http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/ComponentCtrl.html#CE_IMPORTANT)), the event will be queued for later transmission too.
4. If none of the above cases or the widget has no corresponding component on the server, the event will be dropped.

A non-deferred event listener is an event listener (EventListener (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/EventListener.html#>)) that does not implement Deferrable (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Deferrable.html#>). In other words, to minimize the traffic from the client, you might want to implement an event listener with Deferrable (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Deferrable.html#>) if applicable.

Technology Guidelines

ZK provides end-to-end solutions from UI design, development, testing to production. Here are the technology guidelines to help developers to make choices along the way.

If you are new to ZK and prefer to have some prior knowledge of ZK first, you could skip this section and come back later when you understand more about ZK.

MVC vs. MVVM vs. ZScript

They serve different purposes and could work together. However, some developers get confused about these technologies.

When to use MVC and/or MVVM

MVC (Model-View-Control; aka., Model-View-Presenter) and MVVM (Model-View-ViewModel; aka., Presentation Model) are both design patterns that isolate the dependency among the domain data, the domain logic and the user interface. They are both supported by ZK, and they are praised for their separation of concerns^[1] and thus easy to collaborate, develop and maintain. For a production system, it is strongly suggested to take either MVC or MVVM approach.

MVC separates the design into three roles: Model, View, and Controller. The controller is the *middle-man* gluing the view and the model (aka., data).

On the other hand, MVVM has three roles: Model, View, and ViewModel. The View and Model play the same roles as they do in MVC. The ViewModel in MVVM acts like a special controller. Unlike MVC, ViewModel introduces additional abstraction, so it can be written without any detailed knowledge of the view. In other words, the change of the view will have much less impact on the ViewModel. However, the extra abstraction requires extra design thinking.

In most cases, MVVM is suggested for better separation of concerns. On the other hand, MVC is good for small user interfaces and new ZK users, because it is quite straightforward.

When to use zscript

Zscript allows you to embed Java code in ZUML pages. It speeds up the design cycle, so this can be a good approach for prototyping, POC and testing. Zscript is also good for exploiting ZK features and reporting bugs to ZK. However, like any other interpreter, the performance is not very good as it tends to be error-prone. For this reason, it is *not* suggested to use zscript for production systems.

Documentation links

MVC:	<ul style="list-style-type: none">• ZK Developer's Reference: MVC• ZK Developer's Reference: MVVM• ZK Developer's Reference: Performance Tips• Composer^[2] and SelectorComposer^[3]
ZSCRIPT:	<ul style="list-style-type: none">• ZK Developer's Reference: Scripts in ZUML• ZK Studio Essentials: MVC Extractor

Data Binding

When to use

Data Binding automates the data-copy plumbing code (CRUD) between UI components and the data source. It is strongly suggested to use Data Binding whenever applicable because it can help boost programmers' productivity and the code is easy to read and maintain.

When not to use

Barely. However, as Data Binding requires more time and effort to learn than EL expressions, EL expressions provide an alternative for people not familiar with ZK, especially during the UI design phase.

Documentation links

- ZK Developer's Reference: Data Binding

ZUML vs. Richlet vs. JSP

When to use ZUML

ZUML is an XML-based approach to declare UI. It does not require any programming knowledge and it works well with MVC, Data Binding and others. ZUML is strongly suggested for usage unless you have different preferences (such as pure Java and JSP).

However, if most parts of a page are in HTML scripts (such as header and footer) and the UI designer is not familiar with ZUML, you could still use JSP to define the page and then include ZUML page(s) for the part that requires ZK components.

Notice that using ZUML does not prevent you from creating components dynamically in Java. In fact, it is a common practice to use ZUML to layout the theme of a Web application, and then use pure Java to manipulate it dynamically.

When to use Richlet

A richlet is a small Java program that composes a user interface in Java for serving a user's request. You could try to use it if you prefer to compose UI in pure Java (like Swing).

When to use JSP

If you would like to use ZK in legacy JSP pages, you could try one of following approaches:

1. Include <jsp:include> in a ZUML page.
2. Apply ZK JSP Tags^[4] to a JSP page directly.

As described above, if most of a page consists of pure HTML code and the UI designer is not familiar with ZUML, you could use JSP to design the page and then include it in ZUML pages if necessary.

Notice that ZUML supports the use of HTML tags well (without JSP). For more information, please refer to the ZK Developer's Reference: HTML Tags.

Documentation links

ZUML:	<ul style="list-style-type: none"> • ZK Developer's Reference: ZUML • ZK Developer's Reference: HTML Tags
Richlet:	<ul style="list-style-type: none"> • ZK Developer's Reference: Richlet
JSP:	<ul style="list-style-type: none"> • ZK Developer's Reference: Use ZK in JSP and ZK JSP Tags

Bookmarks vs. Multiple Pages

A traditional page-based Web framework forces developers to split an application into pages. On the other hand, Ajax (ZK) allows developers to group a set of functionality into a single desktop-like page that enables a more friendly user experience.

Grouping is much better based on the functionality, unless it is a small application. For example, it might not be a good idea to group administration with, let's say, data entry. Here are some guidelines:

- If a set of functionality is a logical unit to use and/or to develop, you might want to group it into a single page.
- If SEO (i.e., able to be indexed by search engine) is important, it is better to split UI into multiple pages (and turn on the crawlable option).

It does not matter whether the UI shares the same template (such as header and footer) or not because it will be easy anyway to create similar multiple pages (by the use of inclusion, templating and composite).

When to use bookmarks (in single page)

After grouping a set of functionality into a single page, users can still click on the BACK and the FORWARD button to switch among the states of the single page and even bookmark on a particular state, as if there are multiple pages. This can be done by using Browser History Management (aka., bookmarks). You might consider this as a technique to simulate multiple pages (for a single page with multiple states).

When to use multiple pages

If a set of functionality is logically independent of one another, you could make them as separated pages. To jump from one page to another, you could use the so-called send-redirect technique.

Documentation links

Bookmarks:	<ul style="list-style-type: none"> • ZK Developer's Reference: Browser History Management • ZK Developer's Reference: Browser Information and Control
Multiple Pages:	<ul style="list-style-type: none"> • ZK Developer's Reference: Forward and Redirect • ZK Developer's Reference: Include, Templating and Composite for consistent UI across multiple pages.

Native Namespace vs. XHTML Components

ZK provides several ways to use XHTML tags in a ZUML document. Here we will discuss native namespace vs. XHTML components. In a ZUML document, they basically mean the same thing except for the XML namespace. Therefore it should be easy to switch between them.

When to use native namespace

With the use of an XML namespace called the native namespace, you could declare any tags in ZUML as long as they are valid to the client (i.e., any HTML tags for a browser). It is suggested to use this technology if the HTML tags are static. For example, you will not be able to change the content dynamically with Ajax. The header, sidebar, footer and the layout elements are typical examples. It saves the memory on the server.

When to use XHTML components

ZK also provides a set of components to represent each XHTML tag on the server. Unlike the native namespace, these are the 'real' ZK components.

It is suggested that you change their content dynamically because they behave the same as other ZK components. However, since it is a component, it consumes the server's memory.

Documentation links

- ZK Developer's Reference: HTML Tags
- ZK Developer's Reference: Performance Tips|Native vs. XHTML
- ZK Developer's Reference: Performance Tips: Stubonly

Include, Macro, Composite and Templating

They allow developers to modularize the UI such that it becomes easier to develop, maintain and reuse.

When to use include

Include allows you to include a ZUML page, a static page, a JSP page or the result of a servlet. This is the most suitable for usage if you would like to:

1. Include a non-ZUML page
2. Use a page (Page^[5]) to encapsulate a ZUML page^{[6][7]}

The limitation of Include is that you can not encapsulate its behavior in a Java class (like macro or composite components do).

[1] http://en.wikipedia.org/wiki/Separation_of_concerns

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

[3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/select/SelectorComposer.html#>

[4] <http://www.zkoss.org/product/zkjsps.jsp>

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

[6] You have to specify mode="defer" to create a Composer (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Composer.html#>) instance.

[7] Whether a page is required really depends on developer's preference. Introducing a page is more complicated but logically more loosely-coupled.

When to use macro components

Macro components allow developers to define a new component with a ZUML page. So if you would like to reuse a ZUML page across different pages, you can use it because

1. Though optional, you could encapsulate the behavior in a Java class
2. It is easier to map a macro component to another URI, if necessary
3. There is no difference between the use of a macro component and other components

When to use composite components

Composite component is another way to define a new component. With this approach, you could extend a new component from any existent components. However, you must implement a Java class to represent the component^[1]. Unlike macro components, you have to handle the component creation from a ZUML page by yourself^[2].

Feel free to use composite components if you want to inherit the behavior from an existent component, such as Window (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Window.html#>) and Cell (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Cell.html#>), and enhance it to have child components defined in a ZUML document.

[1] Here is an example of composite components in ZK Demo (http://www.zkoss.org/zkdemo/composite/composite_component)

[2] There is a utility called ZK Composite (<https://github.com/zanyking/ZK-Composite>). It allows to define a composite component with Java annotations. Please refer to Small Talks: Define Composite Component using Java Annotation in ZK6 for the details.

When to use templating

Templating allows developers to define UI fragments and define how to assemble them into a complete UI at runtime. Its use is very different from other approaches. Feel free to use templating if you would like the overall layout to be decided at runtime based on, let's say, users' roles or preferences.

Performance and Security

For production systems, it is strongly recommended to take a look at the Performance Tips and Security Tips sections first.

JSF

When to use

JSF is a page-based framework. Because it is too complicated to use, we strongly recommend you to deploy ZK. ZK can do whatever JSF can do or even better. However, if you have to use ZK with legacy JSF, please refer to the Embed ZK Component in Foreign Framework section^[1].

[1] Notice that ZK JSF Components is no longer supported.

Extensions

Here is an overview of the extensions of ZK. They are optional. If you are new to ZK and prefer to have some knowledge of ZK first, you could skip this section and come back later after you understand more about ZK.

There are hundreds of projects which extend ZK's functionality by boosting programmer productivity, providing sample code and many others. For more projects, you could search ZK Project-Info ^[1], Google Code ^[2], Sourceforge.net ^[3], GitHub ^[4], ZK Forge ^[5], etc.

IDE and Tools

ZK Studio ^[6]

ZK Studio is a visual integrated development environment for developing ZK applications with Eclipse IDE ^[7].

REM ^[8]

REM is a NetBeans ^[9] module for ZK. It simplifies the development of ZK with NetBeans IDE ^[9].

ZATS ^[10]

ZATS is a testing tool to automate ZK tests without the need of a browser or a server.

ZK CDT ^[11]

ZK CDT is a component development tool which provides wizards to simplify the creation of ZK components.

ZK Jet

ZK Jet is a browser extension that works with Firefox and Google Chrome. This provides users with a ZK sandbox environment.

run-jetty-run ^[12]

Use this plugin's embedded Jetty distribution to run web applications in Eclipse. This helps to speed up the ZK development by minimizing the deployment time. The project is maintained by Tony Wang ^[13], a member of the ZK Team.

Libraries and Integrations

ZK Spring ^[14]

ZK Spring integrates ZK and Spring framework ^[15]. It supports Spring, Spring Security ^[16], and Spring Web Flow ^[17].

ZK JSP Tags ^[18]

ZK JSP Tags is a collection of JSP tags built upon ZK components, such as that developers could use ZK components and other JSP tags in the same JSP page.

ZKGrails^[19]

ZKGrails is a ZK plugin for the next generation rapid Web development framework, Grails^[20].

ZK addon for Spring ROO^[21]

ZK addon for Spring ROO enables rapid development of ZK / Spring / JPA projects using Spring ROO^[22].

ZK UI Plugin for Grails^[23]

The ZK UI plugin, similar to ZKGrails^[19], can seamlessly integrate ZK with Grails^[20]. It uses the Grails' infrastructures, such as gsp and controllers.

ZEST^[24]

ZEST is a lightweight MVC and REST framework which provides an additional page-level MVC pattern to isolate the request's URI, controller and view (such as ZUML document).

ZK CDI^[25]

ZK CDI is integrated with ZK and JBoss Weld CDI RI^[26].

ZK Seam^[27]

ZK Seam is integrated with ZK and Seam^[28].

ZK JSF Components^[29]

ZK JSF Components are a collection of JSF Components built upon highly interactive ZK Ajax components.

Components and Themes

ZK Themes^[30]

ZK Themes is a collection of various themes, including breeze, silvertail and sapphire.

Keikai (previously ZK Spreadsheet)^[31]

Keikai is a ZK component delivering spreadsheet functionalities like Microsoft Excel to a web application.

ZK Pivottable ^[32]

ZK Pivottable is a ZK component for data summarization that sorts and sums up the original data layout.

ZK Calendar ^[33]

ZK Calendar is a ZK component enabling rich and intuitive scheduling functionality to ZK applications.

ZUSS

ZUSS (ZK User-interface Style Sheet) is an extension to CSS. It is compatible with CSS, while allows the dynamic content, such as variables, mixins, nested rules, expressions, and Java methods with existing CSS syntax.

ZK Incubator Widgets ^[34]

ZK Incubator Widgets ^[35] hosts a collection of incubator widgets, tools and add-ons.

References

- [1] <http://forum.zkoss.org/questions/scope:all/sort:activity-desc/tags:project-info/page:1/>
- [2] <http://code.google.com/query/#q=zk>
- [3] <http://sourceforge.net/search/?q=zk>
- [4] http://github.com/search?langOverride=&q=zk&repo=&start_value=1&type=Repositories
- [5] <http://sourceforge.net/projects/zkforge/>
- [6] <http://www.zkoss.org/product/zkstudio>
- [7] <http://www.eclipse.org>
- [8] <http://rem1.sourceforge.net/>
- [9] <http://www.netbeans.org/>
- [10] <http://www.zkoss.org/product/zats>
- [11] <http://code.google.com/a/eclipselabs.org/p/zk-cdt/>
- [12] <http://code.google.com/p/run-jetty-run/>
- [13] <https://github.com/tony1223>
- [14] <http://www.zkoss.org/product/zkspring>
- [15] <http://www.springsource.org/>
- [16] <http://static.springsource.org/spring-security/site/>
- [17] <http://www.springsource.org/webflow>
- [18] <http://www.zkoss.org/product/zkjsp>
- [19] <http://code.google.com/p/zkgrails/>
- [20] <http://www.grails.org>
- [21] <http://code.google.com/p/zk-roo/>
- [22] <http://www.springsource.org/spring-roo>
- [23] <http://www.grails.org/plugin/zkui>
- [24] <http://code.google.com/p/zest/>
- [25] <http://code.google.com/p/zkcdi/>
- [26] <http://seamframework.org/Weld>
- [27] <http://code.google.com/p/zkseam2/>
- [28] <http://seamframework.org/>
- [29] <http://www.zkoss.org/product/zkjsf>
- [30] <http://code.google.com/p/zkthemes/>
- [31] <http://keikai.io>
- [32] <http://www.zkoss.org/product/zkpivotable>
- [33] <http://www.zkoss.org/product/zkcalendar>
- [34] <http://code.google.com/p/zk-widgets/>
- [35] <https://github.com/jumperchen/zk-widgets-google-code>

Example Project

The example project ^[1] contains the source code mentioned in this book.

References

[1] <https://github.com/zkoss/zkbooks/tree/master/developersreference/developersreference>

UI Composing

Each UI object is represented by a component (Component ^[1]). In this section we will discuss how to declare UI, including XML-based approach and pure-Java approach.

This section describes more general and overall concepts of UI composing. For more detailed and specific topics, please refer to the UI Patterns section. For detailed information on each individual component, please refer to the ZK Component Reference.

References

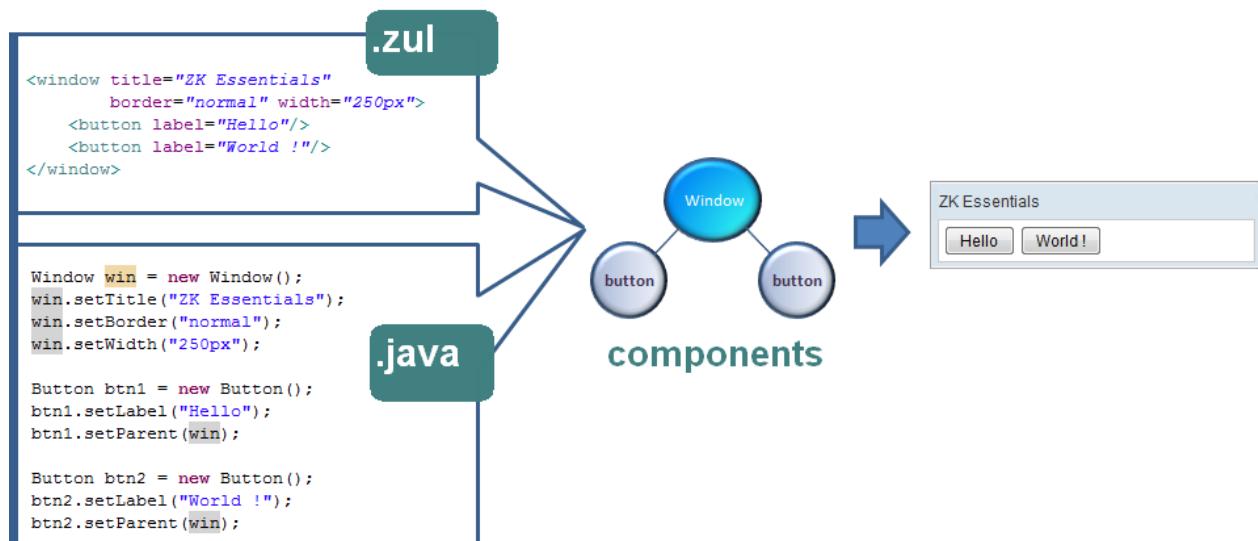
[1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#>

Component-based UI

Overview

Each UI object is represented by a component (Component ^[1]). Thus, composing an UI object is like assembling components. To alter UI one has to modify the states and relationships of components.

For example, as shown below, we declared a Window ^[1] component, enabling the border property to normal and setting its width to a definite 250 pixels. Enclosed in the Window ^[1] component are two Button ^[2] components.



As shown above, there are two ways to declare UI: XML-based approach and pure-Java approach. You can mix them if you like.

Forest of Trees of Components

Like in a tree structure, a component has at most one parent, while it might have multiple children.

Some components accept only certain types of components as children. Some do not allow to have any children at all. For example, Grid^[3] in XUL accepts Columns^[4] and Rows^[5] as children only.

A component without any parents is called a **root component**. Each page is allowed to have multiple root components, even though this does not happen very often.

Notice that if you are using ZUML, there is an XML limitation, which means that only one document root is allowed. To specify multiple roots, you have to enclose the root components with the zk tag. zk tag is a special tag that does not create components. For example,

```
<zk>
  <window/> <!-- the first root component -->
  <div/> <!-- the second root component -->
</zk>
```

getChildren()

Most of the collections returned by a component, such as Component.getChildren()^[6], are live structures. It means that you can add, remove or clear a child by manipulating the returned list directly. For example, to detach all children, you could do it in one statement:

```
comp.getChildren().clear();
```

It is equivalent to

```
for (Iterator it = comp.getChildren().iterator(); it.hasNext();) {
    it.next();
    it.remove();
}
```

Note that the following code will never work because it would cause ConcurrentModificationException.

```
for (Iterator it = comp.getChildren().iterator(); it.hasNext();) {
    ((Component)it.next()).detach();
```

Sorting the children

The following statement will fail for sure because the list is live and a component will be detached first before we move it to different location.

```
Collections.sort(comp.getChildren());
```

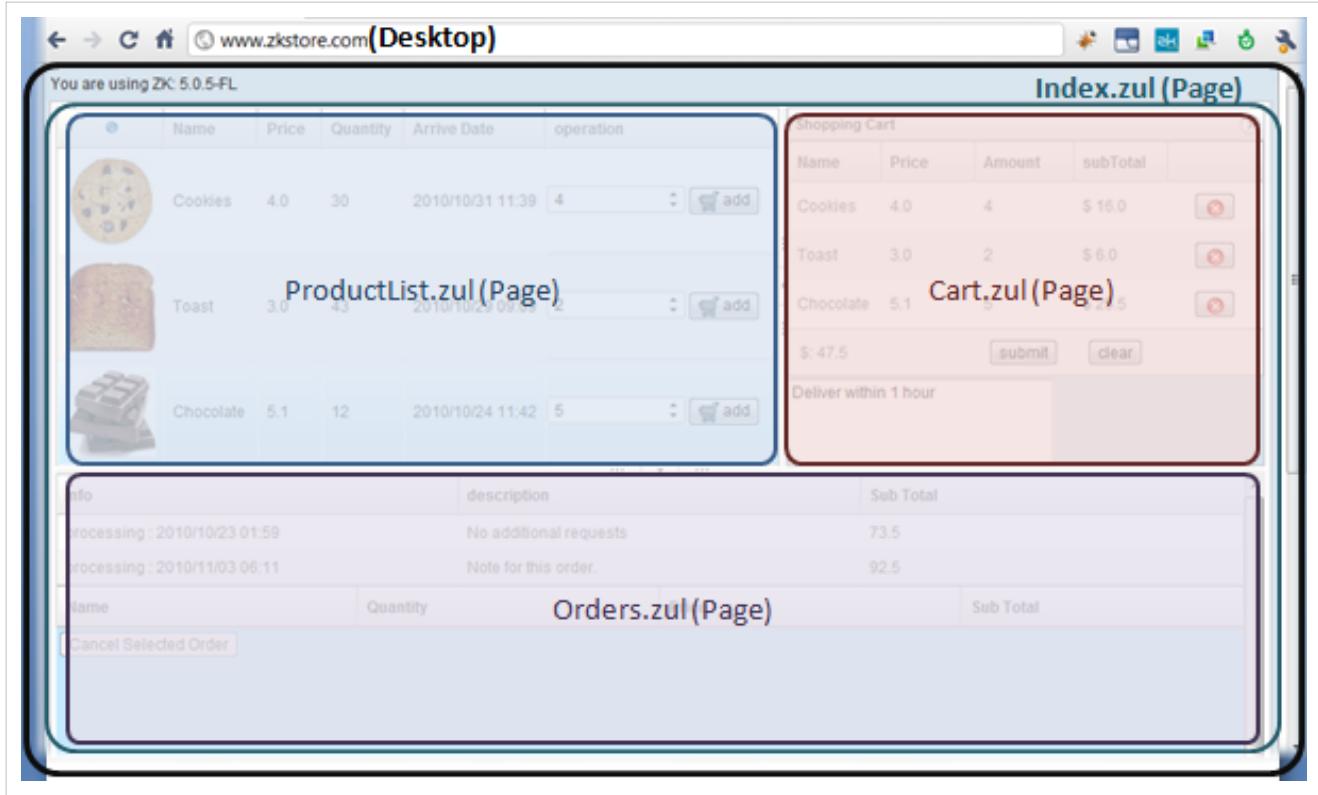
More precisely, a component has at most one parent and it has only one spot in the list of children. It means, the list is actually a set (no duplicate elements allowed). On the other hand, Collections.sort() cannot handle a set correctly.

Thus, we have to copy the list to another list or array and then sort it. java.util.Comparator) Components.sort(java.util.List, java.util.Comparator)^[7] is a utility to simplify the job.

Desktop, Page and Component

A page (Page^[8]) is a collection of components. It represents a portion of the browser window. Only components attached to a page are available at the client. They are removed when they are detached from a page.

A desktop (Desktop^[9]) is a collection of pages. It represents a browser window (a tab or a frame of the browser)^[10]. You might image a desktop representing an independent HTTP request.



A desktop is also a logic scope that an application can access in a request. Each time a request is sent from the client, it is associated with the desktop it belongs. The request is passed to boolean) DesktopCtrl.service(org.zkoss.zk.au.AuRequest, boolean)^[11] and then forwarded to boolean) ComponentCtrl.service(org.zkoss.zk.au.AuRequest, boolean)^[12]. This also means that the application can not access components in multiple desktops at the same time.

Both a desktop and a page can be created automatically when ZK Loader loads a ZUML page or calls a richlet (Richlet.service(org.zkoss.zk.ui.Page))^[13]. The second page is created when the Include^[14] component includes another page with the defer mode. For example, two pages will be created if the following is visited:

```
<!-- the main page -->
<window>
  <include src="another.zul" mode="defer"/> <!-- creates another page -->
</window>
```

Notice that if the mode is not specified (i.e., the instant mode), Include^[14] will not be able to create a new page. Rather, it will append all components created by another.zul as its own child components. For example,

```
<window>
  <include src="another.zul"/> <!-- default: instant mode -->
</window>
```

is equivalent to the following (except div is not a space owner, see below)

```
<window>
  <div>
    <zscript>
      execution.createComponents("another.zul", self, null);
    </zscript>
  </div>
</window>
```

-
- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Window.html#>
 - [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Button.html#>
 - [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Grid.html#>
 - [4] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Columns.html#>
 - [5] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Rows.html#>
 - [6] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#getChildren\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#getChildren())
 - [7] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Components.html#sort\(java.util.List,](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Components.html#sort(java.util.List,)
 - [8] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Page.html#>
 - [9] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Desktop.html#>
 - [10] Under portal environment, there might be multiple desktops in one browser window. However, it is really important in the developer's viewpoint.
 - [11] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/DesktopCtrl.html#service\(org.zkoss.zk.au.AuRequest,](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/DesktopCtrl.html#service(org.zkoss.zk.au.AuRequest,)
 - [12] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/ComponentCtrl.html#service\(org.zkoss.zk.au.AuRequest,](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/ComponentCtrl.html#service(org.zkoss.zk.au.AuRequest,)
 - [13] [>http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Richlet.html#service\(org.zkoss.zk.ui.Page\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Richlet.html#service(org.zkoss.zk.ui.Page))
 - [14] [>http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Include.html#](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Include.html#)

Attach a Component to a Page

A component is available at the client only if it is attached to a page. For example, the window created below will not be available at the client.

```
Window win = new Window();
win.appendChild(new Label("foo"));
```

A component is a POJO object. If you do not have any reference to it, it will be recycled when JVM starts garbage collection ([http://en.wikipedia.org/wiki/Garbage_collection_\(computer_science\)](http://en.wikipedia.org/wiki/Garbage_collection_(computer_science))).

There are two ways to attach a component to a page:

1. Append it as a child of another component that is already attached to a page (
[Component.appendChild\(org.zkoss.zk.ui.Component\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#appendChild(org.zkoss.zk.ui.Component, org.zkoss.zk.ui.Component)) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#appendChild\(org.zkoss.zk.ui.Component, org.zkoss.zk.ui.Component\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#appendChild(org.zkoss.zk.ui.Component, org.zkoss.zk.ui.Component))), [org.zkoss.zk.ui.Component](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#insertBefore(org.zkoss.zk.ui.Component, org.zkoss.zk.ui.Component)))
[Component.insertBefore\(org.zkoss.zk.ui.Component, org.zkoss.zk.ui.Component\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#insertBefore(org.zkoss.zk.ui.Component, org.zkoss.zk.ui.Component)) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#insertBefore\(org.zkoss.zk.ui.Component, org.zkoss.zk.ui.Component\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#insertBefore(org.zkoss.zk.ui.Component, org.zkoss.zk.ui.Component))), or
[Component.setParent\(org.zkoss.zk.ui.Component\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#setParent(org.zkoss.zk.ui.Component)) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#setParent\(org.zkoss.zk.ui.Component\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#setParent(org.zkoss.zk.ui.Component)))).
2. Invoke [Component.setPage\(org.zkoss.zk.ui.Page\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#setPage(org.zkoss.zk.ui.Page)) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#setPage\(org.zkoss.zk.ui.Page\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#setPage(org.zkoss.zk.ui.Page))) to attach it to a page directly. It is also another way to make a component become a root component.

Since a component can have at most one parent and be attached at most one page, it will be detached automatically from the previous parent or page when it is attached to another component or page. For example, `b` will be a child of `win2` and `win1` has no child at the end.

```
Window win1 = new Window();
Button b = new Button();
win1.appendChild(b);
```

```
win2.appendChild(b); //implies detach b from win1
```

Detach a Component from a Page

To detach a Component from the page, you can either invoke `comp.setParent(null)` if it is not a root component or `comp.setPage(null)` if it is a root component. `Component.detach()` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/zk/ui/Component.html#detach\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/zk/ui/Component.html#detach())) is a shortcut to detach a component without knowing if it is a root component.

Invalidate a Component

When a component is attached to a page, the component and all of its descendants will be rendered. On the other hand, when a state of an attached component is changed, only the changed state is sent to client for update (for better performance). Very rare, you might need to invoke `Component.invalidate()` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#invalidate\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#invalidate())) to force the component and its descendants to be rerendered^[1].

There are only a few reasons to invalidate a component, but it is still worthwhile to note them down:

1. If you add more than 20 child components, you could invalidate the parent to improve the performance. Though the result Ajax response might be longer, the browser will be more effective in replacing a DOM tree rather than adding DOM elements.
2. If a component has a bug that does not update the DOM tree correctly, you could invalidate its parent to resolve the problem^[2].

[1] ZK Update Engine will queue the *update* and *invalidate* commands, and then optimize them before sending them back to the client (for better performance)

[2] Of course, remember to let us know and we will fix it in the upcoming version.

Don't Cache Components Attached to a Page in Static Fields

As described above, a desktop is a logical scope which can be accessed by the application when serving a request. In other words, the application cannot detach a component from one desktop to another desktop. This typically happens when you cache a component accidentally.

For example, the following code will cause an exception if it is loaded multiple times.

```
<window apply="foo.Foo"/> <!-- cause foo.Foo to be instantiated and executed -->
```

and `foo.Foo` is defined as follows^[1].

```
package foo;
import org.zkoss.zk.ui.*;
import org.zkoss.zul.*;

public class Foo implements org.zkoss.zk.util.Composer {
    private static Window main; //WRONG! don't cache it in a static field
    public void doAfterCompose(Component comp) {
        if (main == null)
            main = new Window();
        comp.appendChild(main);
    }
}
```

The exception is similar to the following:

```
org.zkoss.zk.ui.UiException: The parent and child must be in the same
desktop: <Window u1EP0>

org.zkoss.zk.ui.AbstractComponent.checkParentChild(AbstractComponent.java:1057)

org.zkoss.zk.ui.AbstractComponent.insertBefore(AbstractComponent.java:1074)
    org.zkoss.zul.Window.insertBefore(Window.java:833)

org.zkoss.zk.ui.AbstractComponent.appendChild(AbstractComponent.java:1232)
    foo.Foo.doAfterCompose(Foo.java:10)
```

[1] A composer (Component (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#>)) is a controller that can be associated with a component for handling the UI in Java. For the information, please refer to the Composer section.

Component Cloning

All components are cloneable (java.lang.Cloneable). It is simple to replicate components by invoking Component.clone() ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#clone\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#clone())).

```
main.appendChild(listbox.clone());
```

Notice

- It is a *deep clone*. That is, all children and descendants are cloned too.
- The component returned by Component.clone() ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#clone\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#clone())) does not belong to any pages. It doesn't have a parent either. You have to attach it manually if necessary.
- ID, if any, is preserved. Thus, you *cannot* attach the returned component to the same ID space without modifying ID if there is any.

Similarly, all components are serializable (java.io.Serializable). Like cloning, all children and descendants are serialized. If you serialize a component and then de-serialize it back, the result will be the same as invoking Component.clone() ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#clone\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#clone()))^[1].

[1] Of course, the performance of Window (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Window.html#>) is much better.

ID Space

ID Space

It is common to decompose a visual presentation into several subsets or ZUML pages. For example, you may use a page to display a purchase order, and a modal dialog to enter the payment term. If all components are uniquely identifiable in the same desktop, developers have to maintain the uniqueness of all identifiers for all pages that might be created in the same desktop. This step can be tedious, if not impossible, for a sophisticated application.

The concept of ID space is hence introduced to resolve this issue. An ID space is a subset of components of a desktop. The uniqueness is guaranteed only in the scope of an ID space. Thus, developers could maintain the subset of components separately without the need to worry if there are any conflicts with other subsets.

Window (Window^[1]) is a typical component that is an ID space. All descendant components of a window (including the window itself) form an independent ID space. Thus, you could use a window as the topmost component to group components. This way developers only need to maintain the uniqueness of each subset separately.

By and large, every component can form an ID space as long as it implements IdSpace^[1]. This type of component is called the space owner of the ID space after the component is formed. Components in the same ID space are called "fellows".

When a page implements IdSpace^[1], it becomes a space owner. In addition, the macro component and the include component (Include^[14]) can also be space owners.

Another example is idspace (Idspace^[2]). It derives from div, and is the simplest component implementing IdSpace^[1]. If you don't need any feature of window, you could use idspace instead.

You could make a standard component as a space owner by extending it to implement IdSpace^[1]. For example,

```
public class IdGrid extends Grid implements IdSpace {
    //no method implementation required
}
```

Tree of ID Space

If an ID space has a child ID space, the components of the child space are not part of the parent ID space. But the space owner of the child ID space will be an exception in this case. For example, if an ID space, let's say X, is a descendant of another ID space, let's say Y, then space X's owner is part of space Y. However, the descendants of X is not a part of space Y.

For example, see the following ZUML page

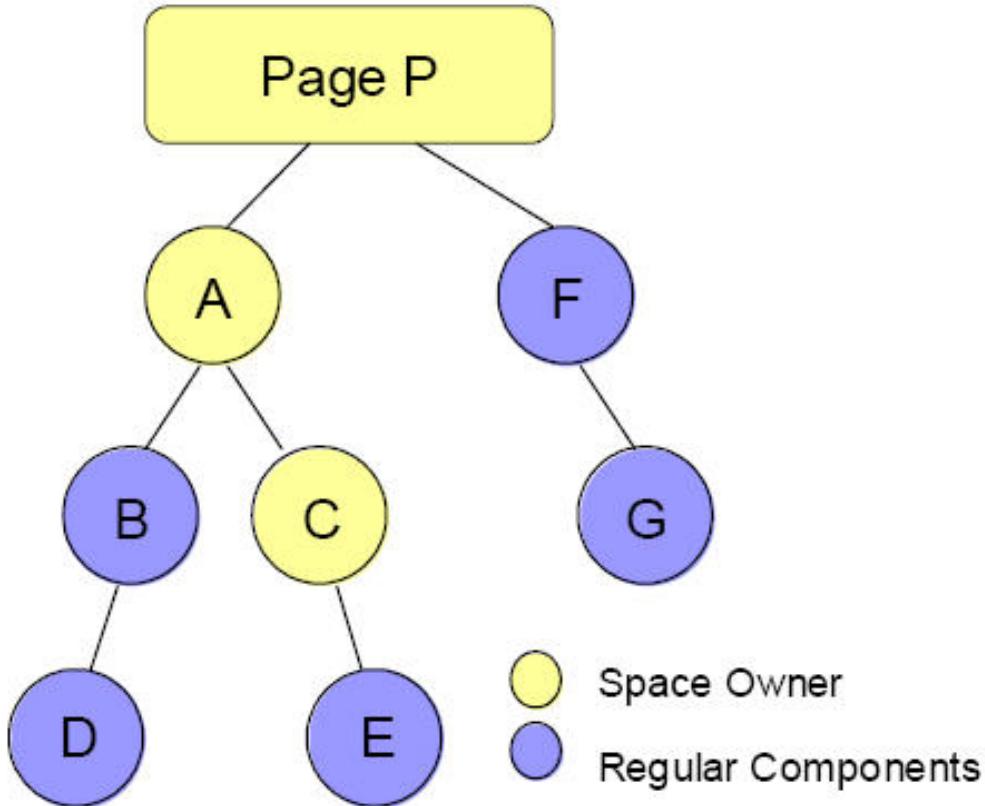
```
<?page id="P"?>
<zkl>
    <window id="A">
        <hbox id="B">
            <button id="D" />
        </hbox>
        <window id="C">
            <button id="E" />
        </window>
    </window>
    <hbox id="F">
```

```

<button id="G" />
</hbox>
</zk>

```

will form ID spaces as follows:



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

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

getFellow and getSpaceOwner

The owner of an ID space could be retrieved by Component.getSpaceOwner()^[3] and any components in an ID space could be retrieved by Component.getFellow(java.lang.String)^[4], if it is assigned with an ID (Component.setId(java.lang.String)^[5]).

Notice that the `getFellow` method can be invoked against any components in the same ID space, not just the space owner. Similarly, the `getSpaceOwner` method returns the same object for any components in the same ID space, no matter if it is the space owner or not. In the example above, if C calls `getSpaceOwner` it will get C itself, if C calls `getSpaceOwnerOfParent` it will get A.

Composer and Fellow Auto-wiring

With ZK Developer's Reference/MVC, you generally don't need to look up fellows manually. Rather, they could be *wired* automatically by using the auto-wiring feature of a composer. For example,

```
public class MyComposer extends SelectorComposer {
    @Wire
    private Textbox input; //will be wired automatically if there is a
fellow named input

    public void onOK() {
        Messsagebox.show("You entered " + input.getValue());
    }

    public void onCancel() {
        input.setValue("");
    }
}
```

Then, you could associate this composer to a component by specifying the apply attribute as shown below.

```
<window apply="MyComposer">
    <textbox id="input"/>
</window>
```

Once the ZUML document above is rendered, an instance of MyComposer will be instantiated, and the `input` member will also be initialized with the fellow named `input`. This process is called "auto-wiring". For more information, please refer to the Wire Components section.

Find Component Manually

There are basically two approaches to look for a component: by use of CSS-like selector and file system-like path. The CSS-like selector is more powerful and suggested if you're familiar with CSS selectors, while a filesystem-like path is recommended if you're familiar with the filesystem's path.

Selector

`Component.query(java.lang.String)`^[6] and `Component.queryAll(java.lang.String)`^[7] are the methods to look for a component by use of CSS selectors. For example,

```
comp.query("#ok"); //look for a component whose ID is ok in the same ID
space
comp.query("window #ok"); //look for a window and then look for a
component with ID=ok in the window
comp.queryAll("window button"); //look for a window and then look for
all buttons in the window
```

`Component.query(java.lang.String)`^[6] returns the first matched component, or null if not found. On the other hand, `Component.queryAll(java.lang.String)`^[7] returns a list of all matched components.

Path

ZK provides a utility class called Path^[8] to simplify the location of a component among ID spaces. The way of using it is similar to `java.io.File`. For example,

The formal syntax of the paths

```
/ [/] SPACE_OWNER_ID / [SPACE_OWNER_ID...] / FELLOW_ID
```

- The last element is a fellow (component) ID.
- page ID should start with double slash //

For example:

```
// Two different ways to get the same component E
Path.getComponent("/A/C/E"); // if call Path.getComponent under the same
page.

new Path("//A/C", "E").getComponent(); // the same as new
Path("//A/C/E").getComponent()

// B and D are fellows in the Id space of A
Path.getComponent("//A/B"); // get B
Path.getComponent("//A/D"); // get D
```

Different Page

If a component belongs to another page, we can retrieve it by starting with the page's ID. Notice that double slashes have to be specified in front of the page's ID.

```
Path.getComponent("//P/A/C/E"); // for page, you have to use // as prefix
```

Notice that the page's ID can be assigned with the use of the page directive as follows.

```
<?page id="foo"?>
<window/>
```

UUID

A component has another identifier called UUID (Universal Unique ID). It is assigned automatically when the component is attached to a page. UUID of a component is unique in the whole desktop (if it is attached).

Application developers rarely need to access it.

In general, UUID is independent of ID. UUID is assigned automatically by ZK, while ID is assigned by the application. However, if a component implements RawId^[9], ID will become UUID if the application assigns one. Currently, only components from the XHTML component set implements RawId^[9].

Version History

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

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/IdSpace.html#>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Idspace.html#>
- [3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#getSpaceOwner\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#getSpaceOwner())
- [4] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#getFellow\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#getFellow(java.lang.String))
- [5] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#setId\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#setId(java.lang.String))
- [6] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#query\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#query(java.lang.String))
- [7] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#queryAll\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#queryAll(java.lang.String))
- [8] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Path.html#>
- [9] <http://www.zkoss.org/javadoc/latest/zk/org/zkos/zk/ui/ext/RawId.html#>

ZUML

There are two ways to compose UI: XML-based approach and pure-Java approach. Here we will describe XML-based approach. For pure-Java approach, please refer to the next chapter.

The declaration language is called ZK User Interface Markup Language (ZUML). It is based on XML. Each XML element instructs ZK Loader to create a component. Each XML attribute describes what value to be assigned to the created component. Each XML processing instruction describes how to process the whole page, such as the page title. For example,

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

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

Auto-completion with Schema

When working with a ZUML document, it is suggested to use ZK Studio ^[1] since it provides a lot of features to simplify editing, such as *content assist* and *visual editor*.

If you prefer not to use ZK Studio, you could specify the XML schema in a ZUML document as shown below. Many XML editors work better, such as when with auto-complete, if XML schema is specified correctly.

```
<window xmlns="http://www.zkoss.org/2005/zul"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.zkoss.org/2005/zul
    http://www.zkoss.org/2005/zul/zul.xsd">
```

The ZUL schema can be downloaded from <http://www.zkoss.org/2005/zul/zul.xsd> ^[2]. In addition, you can find `zul.xsd` under the `dist/xsd` directory in the ZK binary distribution.

This section is about the general use of ZUML. For a complete reference, please refer to ZUML Reference.

References

- [1] <http://www.zkoss.org/product/zkstudio.dsp>
- [2] <http://www.zkoss.org/2005/zul/zul.xsd>

XML Background

Overview

This section provides the most basic concepts of XML to work with ZK. If you are familiar with XML, you could skip this section. If you want to learn more, there are a lot of resources on Internet, such as http://www.w3schools.com/xml/xml_whatis.asp^[1] and <http://www.xml.com/pub/a/98/10/guide0.html>^[2].

XML is a markup language much like HTML but with stricter and cleaner syntax. It has several characteristics worthwhile to take notes of.

Document

The whole XML content, no matter whether it is in a file or as a string, is called an XML document.

Character Encoding

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

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

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

Elements

An XML element is everything from (including) the element's start tag to (including) the element's end tag.

An element can contain other elements, let it be simple text or a mixture of both. Elements can also have attributes. For example,

```
<window title="abc">
  <button label="click me"/>
</window>
```

where both window and button are elements, while title is an attribute of the window element. The button element is nested in the window element. We call the window component the parent element of button, while the button component is a child element of the window.

The document root is the topmost element (without any parent element). There is exactly one document root per XML document.

Elements Must Be Well-formed

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

Description	Code
Close by an end tag:	<code><window></window></code>
Close without an end tag:	<code><window/></code>

Second, elements must be properly nested.

Result	Code
Correct:	<code><window> <groupbox> Hello World! </groupbox> </window></code>
Wrong:	<code><window> <groupbox> Hello World! </window> </groupbox></code>

XML treats every tag as a node in a tree. A node without a parent node is a root component, and it is the root of a tree. In each zul file, only **ONE** tree is allowed.

For example, for being a whole zul file, the following is allowed as it has only one root component.

```
<button/>
```

And for being a whole zul file, the following is not allowed as it has more than one root component.

```
<button/>
<button/>
```

You can solve the problem simply by adding a tag to enclose the whole zul file to serve as the parent node, so that the zul file has one single tree again.

```
<window>
    <button />
    <button />
</window>
```

Special Characters Must Be Replaced

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

Special Character	Replaced With	
<	<	
>	>	
&	&	
"	"	
'	'	
\t (TAB)			Required only if used in an XML attribute's value
\n (Linefeed)	
	Required only if used in an XML attribute's value

Alternatively, you could tell XML parser not to interpret a piece of text by using CDATA. See the following:

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

It is suggested to always add `<! [CDATA[]]>` inside your `<zscript> </zscript>`. Thus you don't have to worry about the escape sequences for special characters like "&", "<". In addition, the code can also become much easier to read and maintain.

Attribute Values Must Be Specified and Quoted

Result	Code
Correct:	<code>width="100%"</code> <code>checked="true"</code>
Wrong:	<code>width=100%</code> <code>checked</code>

Both the single quote ('') and the double quote ("") can be used, so if the value has double quotes, you could use the single quote to enclose it. For example,

```
<button onClick='alert("Hello, There")' />
```

Of course, you can always use " to denote a double quote.

Comments

A comment is used to leave a note or to temporarily disable a block of XML code. To add a comment in XML, use <!-- and --> to mark the comment body.

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

Processing Instruction

A processing instruction is used to carry out the instruction to the program that processes the XML document. A processing instruction is enclosed with <? and ?>. For example,

```
<?page title="Foo"?>
```

Processing instructions may occur anywhere in an XML document. However, most ZUML processing instructions must be specified at the topmost level (the same level as the document root).

References

- [1] http://www.w3schools.com/xml/xml_whatis.asp
- [2] <http://www.xml.com/pub/a/98/10/guide0.html>

Basic Rules

If you are not familiar with XML, please take a look at XML Background first.

An XML Element Represents a Component

Each XML element represents a component, except for special elements like <zk> and <attribute>. Thus, the following example will cause three components (window, textbox and button) to be created when ZK Loader processes it.

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

In addition, the parent-child relationship of the created components will follow the same hierarchical structure as the XML document. In the previous example, window will be the parent of textbox and button, while textbox is the first child and button is the second.

Special XML Elements

There are a few elements dedicated to special functionality rather than a component. For example,

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

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

```
<window>
  <zk if="${whatever}">
    <textbox/>
    <textbox/>
  </zk>
</window>
```

is equivalent to

```
<window>
  <textbox if="${whatever}" />
  <textbox if="${whatever}" />
</window>
```

For more information about special elements, please refer to ZUML Reference.

An XML Attribute Assigns a Value to a Component's Property or Event Listener

Each attribute, except for special attributes like if and forEach, represents a value that should be assigned to a property of a component after it is created. The attribute name is the property name, while the attribute value is the value to assign. For example, the following example assigns "Hello" to the window's title property. More precisely, Window.setTitle(java.lang.String)^[1] will be called with the argument "Hello".

```
<window title="Hello"/>
```

Like JSP, you could use EL for the value of any attributes. The following example assigns the value of the request parameter called name to window's title.

```
<window title="${param.name}" />
```

For more information about EL expressions, please refer to ZUML Reference.

Assign Event Listener if the Name Starts With on

If the attribute name starts with on and the third letter is uppercase, an event listener is assigned. For example, we can register an event listener to handle the onClick event as follows:

```
<button onClick="do_something_in_Java()" />
```

The attribute value must be a valid Java code, and it will be interpreted^[2] when the event is received. You could specify different languages by prefixing the language name. For example, we could write the event listener in Groovy as follows.

```
<vlayout onClick="groovy:self.appendChild(new Label('New'));">
Click me!
</vlayout>
```

- [1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Window.html#setTitle\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Window.html#setTitle(java.lang.String))
[2] ZK uses BeanShell (<http://www.beanshell.org>) to interpret it at run time

Special Attributes

There are a few special attributes dedicated to special functionality rather than assigning properties or handling events. For example, the `forEach` attribute is used to specify a collection of objects such that the XML element it belongs will be evaluated repeatedly for each object of the collection.

```
<listbox>
  <listitem forEach="${customers}" label="${each.name}" />
</listbox>
```

For more information about special attributes, please refer to the Iterative Evaluation section and the ZUML Reference

An XML Text Represents Label Component or Property's Value

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

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

is equivalent to

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

An XML Text as Property's Value

Depending on the component's implementation, the text nested in an XML element can be interpreted as the value of a component's particular property. For example, `Html` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Html.html#>) is one of these kinds of components, and

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

is equivalent to

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

This is designed to make it easy to specify multiple-line value. This is usually used by a particular component that requires a multiple-lines value. For a complete list of components that interpret the XML text as a property's value, please refer to the ZUML Reference.

An XML Processing Instruction Specifies the Page-wide Information

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

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

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

The other directives include the declaration of components, the class for initializing a page, the variable resolver for EL expressions, and so on. For more information about directives, please refer to ZUML Reference.

EL Expressions

Overview

EL expressions are designed to make a ZUML document easier to access objects available in the application, such as the application data and parameters. For a complete introduction, please refer to ZUML Reference/EL Expressions.

An EL expression is an expression enclosed with \${ } and , i.e., the syntax \${expr}. For example,

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

When an EL expression is used as an attribute value, it could return any kind of objects as long as the attribute allows. For example, the following expressions will be evaluated to boolean and int respectively.

```
<window if="${some > 10}"><!-- boolean -->
<progressmeter value="${progress}" /><!-- integer -->
```

If the class does not match, ZK Loader will try to coerce it to the correct one. If a failure has occurred, an exception is thrown.

Multiple EL expressions could be specified in a single attribute:

```
<window title="${foo.name}: ${foo.version}">
```

Example

EL Expression	Result
<code> \${1 > (4/2)}</code>	false
<code> \${100.0 == 100}</code>	true
<code> \${'a' < 'b'}</code>	true
<code> \${'hip' gt 'hit'}</code>	false
<code> \${1.2E4 + 1.4}</code>	12001.4
<code> \${3 div 4}</code>	0.75
<code> \${10 mod 4}</code>	2
<code> \${empty param.add}</code>	true if the request parameter named add is null or an empty string

\${param['mycom.productId']}	The value of the request parameter named mycom.productId
------------------------------	--

- The example is from JSP Tutorial ^[1].
- For more information please refer to Operators and Literals.

Difference from Java

- A string can be enclosed with either single quotes or double quotes. In other words, 'abc' is equivalent to "abc".
- The empty operator is useful for testing null and empty string, list and map, such as \${empty param.add}.
- The . operator can be used to access a property of an object (assuming that there is a get method of the same name) or a value of a map, such as \${foo.value.name}.
- The [] operator can be used to access an item of a list or array, a value of a map, and a property of an object (assuming that there is a get method of the same name), such as \${ary[5]} and \${wnd['title']}.
- null is returned if the value is not found and the index is out-of-bound.

For more information please refer to Operators and Literals.

Resolving EL Variables

EL expressions are evaluated on the server when the page is rendered. Thus, an EL variable can access:

- Components by using its ID
- Variables defined in zscript
- Implicit objects
- Scoped attributes

```

<!-- self is an implicit object referring to the component itself -->
<textbox id="tb" value="${self.parent.title}" />

<!-- tb, the ID of a textbox, is the object reference of the textbox component -->
${tb.value}

<!-- param is an implicit object -->
<button label="Enter" if="${not empty param.edit}" />

<zscript><![CDATA[
    Date now = new Date();
]]></zscript>
<!-- now is a variable defined in zscript -->
<datebox value="${now}" />
```

Resolving Order

ZK resolves a variable from smaller scope to larger scope in the order below:

1. zscript variable
2. execution
3. component
4. page
5. desktop
6. session
7. application

Hence, if there is an attribute value in a smaller scope, it will shadow the same attribute in the larger scope.

```
<div id="parent">
    <zscript><! [CDATA[
        //the smaller scope (lower one) can shadow the upper one
        application.setAttribute("myname", "in application");
        session.setAttribute("myname", "in session");
        desktop.setAttribute("myname", "in desktop");
        page.setAttribute("myname", "in page");
        parent.setAttribute("myname", "in component");
        execution.setAttribute("myname", "in execution");
    ]]></zscript>
    Resolved result:
    <label style="font-weight: bold" value="${myname}" />
</div>
```

In the example above, the resolved result is **in execution**. But if you remove line 9, you will see **in component**.

(check org.zkoss.zk.xel.impl.ExecutionResolver.resolveVariable0())

Furthermore, you could define a variable resolver to associate a name with an object or map a function to a Java static method as described in the following.

Variable Resolver

If you would like to support many variables, you could implement a variable resolver: a class that implements VariableResolver^[2].

```
package foo;
public class CustomerResolver implements org.zkoss.xel.VariableResolver
{
    public Object resolveVariable(String name) {
        if ("customers".equals(name))
            return Customer.getAll("*");
        // if ("recent".equals(name))
        //     return something_else;
        return null; //not a recognized variable
    }
}
```

Then, you could specify it in a variable-resolver directive, such as:

```
<?variable-resolver class="foo.CustomerResolve"?>

<listbox>
    <listitem label="${each.name}" forEach="${customers}"/>
</listbox>
```

System-level Variable Resolver

If you have a variable resolver that will be used on every page, you can register a system-level variable resolver rather than specifying it on every page.

This can be done by specifying a variable resolver you have implemented in `WEB-INF/zk.xml` as follows. For more information, please refer to ZK Configuration Reference.

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

Then, when a page is created each time, an instance of the specified class will be instantiated and registered as if it is specified in the variable-resolver element.

Notice that since a new instance of the variable resolver is created on each page, there will not be any concurrency issues.

Calling Java Methods

Define in Page-Scope

The collection object could be retrieved by invoking a static method. For example, suppose that we have a class and a static method as follows:

```
package foo;
public class Customer {
    public static Collection<Customer> getAll(String condition) {
        //...returns a collection of customers
    }
    public String getName() {
        return _name;
    }
    //...
}
```

Then, we could retrieve them with the `xel-method` directive:

```
<?xel-method prefix="c" name="getAllCustomers" class="foo.Customer"
    signature="java.util.Collection getAll(java.lang.String)"?><!-- Generics not allowed -->
<listbox>
    <listitem label="${each.name}" forEach="${c:getAllCustomers('*')}"/>
</listbox>
```

Define as a Tag Library

If you have several static methods, you can declare them in an XML file called taglib, such as

```
<taglib>
  <function>
    <name>getAllCustomers</name>
    <function-class>foo.Customer</function-class>
    <function-signature>
      java.util.Collection getAll(java.lang.String)
    </function-signature>
    <description>
      Returns a collection of customers.
    </description>
  </function>
  <!-- any number of functions are allowed -->
</taglib>
```

Then, you can use them by specifying it in a taglib directive.

```
<?taglib uri="/WEB-INF/tld/my.tld" prefix="my"?>
<listbox>
  <listitem label="${each.name}" forEach="${my:getAllCustomers('*')}"/>
</listbox>
```

EL 3.0 Support

Since ZK 8, ZK supports some syntaxes of Java EE 7 Expression Language 3, see examples ^[3].

References

- [1] <http://download.oracle.com/javaee/1.4/tutorial/doc/JSPIntro7.html>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xel/VariableResolver.html#>
- [3] http://books.zkoss.org/zk-mvvm-book/9.5/data_binding/el_expression.html

Scripts in ZUML

Embed Server-side Script Code

To make it easier to create a dynamic web page, the ZUML document allows you to embed the script code. Notice that there are two types of script code: server-side and client-side. How the client-side code can be embedded is discussed in the Client-side UI Composing and Client-side Event Listening sections. Here we will discuss how to embed the server-side script code in a ZUML document.

Fast Prototyping

Embedding Java code in a ZUML page is a powerful way for fast prototyping. For example, you can quickly build a prototype UI page to discuss with business analysts and UI designers. Then, modify it directly and get feedback immediately without going through drawings and even recompiling.

Performance Notice

Notice that the performance of BeanShell is not good and, like any interpreter, typos can be found only when it is evaluated. For more information, please refer to the Performance Tips section

2 Places to Embed

Depending on the requirement, there are two ways to embed the server-side script code in a ZUML document: the `zscript` element and the event handler. The `zscript` element is used to embed the code that will execute when the page is loaded, while the event handler will execute when the event is received.

zscript

First, you could embed the code inside the `zscript` element, such that they will be evaluated when the page is rendered^[1]. For example,

```
<zscript>
//inside is zscript
//you can declare variable, function, and even Java class here.
void foo(String msg) {
    //...
}
comp.addEventListener("onClick",
    new EventListener() {
        public void onEvent(Event event) {
            //...
        }
    });
</zscript>
```

Notice that, by default, the code inside the `zscript` element is Java but you could also use other languages, such as Groovy. Keep in mind that it is *interpreted* at run time (by Beanshell^[2]), so typo or syntax error will be found only when it is interpreted. In addition, it runs on the server, so it could access any Java libraries. You could even define variables, methods, and classes with it, and they are visible to EL expressions on the same page.

CDATA

The code embedded in the zscript element must be a valid XML text. In other words, you must encode the special characters well, such as < must be replaced with <, & with & and so on. In addition to encoding individual characters, you can also enclose the whole code with XML CDATA as follows.

```
<zscript><! [CDATA[
if (some < another && another < last) //OK since CDATA is used
    doSomething();
]]></zscript>
```

As depicted CDATA is represented with <! [CDATA[and]]>.

-
- [1] The zscript element has an attribute called deferred that could make the evaluation as late as possible
 - [2] <https://github.com/beanshell/beanshell>

Class Declaration

You could define a class declared in a ZUML document, and the class is accessible only in the page it was defined. For example,

```
<?xml version="1.0" encoding="UTF-8"?>
<zk>
<zscript><! [CDATA[
public class FooModel extends AbstractTreeModel {
    public FooModel() {
        super("Root");
    }
    public boolean isLeaf(Object node) {
        return getLevel((String)node) >= 4; //at most 4 levels
    }
    public Object getChild(Object parent, int index) {
        return parent + "." + index;
    }
    public int getChildCount(Object parent) {
        return 5; //each node has 5 children
    }
    public int getIndexofChild(Object parent, Object child) {
        String data = (String)child;
        int i = data.lastIndexOf('.');
        return Integer.parseInt(data.substring(i + 1));
    }
    private int getLevel(String data) {
        for (int i = -1, level = 0;; ++level)
            if ((i = data.indexOf('.', i + 1)) < 0)
                return level;
    }
}
FooModel model = new FooModel();
]]></zscript>
<tree model="${model}">
```

```

<treetreecols>
    <treetreecol label="Names"/>
</treetreecols>
</tree>
</zk>

```

Event Handlers

Second, you could put the code inside an event handler, such that it will execute when the event is received, as depicted below.

```
<button onClick='alert("event handler for onXXX inside ZUML is also zscript")' />
```

Notice that the name of the event must start with `on`, and the third letter must be an **upper** case. Otherwise, it will be considered as a property.

Again, the code is Java interpreted at run time and running on the server. For client-side listening, please refer to the Client-side Event Listening section.

For the sake of discussion, we call it zscript no matter the code is embedded in the `zscript` element or in an event handler.

Attribute

If the code is too complicated, you could specify the event handle in the attribute element. For example,

```

<button label="hi">
    <attribute name="onClick"><! [DATA[
        if (anything > best)
            best = anything;
    ]]></attribute>
</button>

```

Distinguish zscript from EL

Keep in mind, an EL expression is enclosed by `{}` .

For example, `${self.label}` and `${ok.label}` are both EL expressions in the following example:

```

<window>
    <button label="ok" id="${self.label}" />
    ${ok.label}
</window>

```

On the other hand, in the following example, `alert(self.label)` is not an EL expression. Rather, it's the zscript code:

```

<window>
    <button label="ok" onClick='alert(self.label)' />
</window>

```

You cannot mix the use of EL expressions with zscript:

```

<window>
    <!-- It's wrong, for java don't accept syntax as ${}-->

```

```
<button label="ok" onClick='alert(${self.label})' />
</window>
```

Also notice that the evaluation of EL expressions is very fast, so EL can be used in a production system. On the other hand, zscript is suggested to use only in prototyping or quick-fix.

Variables Defined in zscript Visible to EL

A variable defined in zscript is visible to EL expression, unless it is a local variable, which will be discussed later.

```
<zscript>
Date now = new Date();
</zscript>
${now}
```

Java Interpreter

The default interpreter is based on BeanShell (<http://www.beanshell.org>). It is a Java Interpreter.

Scope for Each ID Space

The Java interpreter is a *multi-scope* interpreter. It creates a scope for each ID space. Since ID space is hierarchical, so are the scopes. If a variable cannot be found in the current ID space, it will go further to parent's ID space to try to resolve the variable.

For example, in the following example, two logical scopes are created for window^[1] A and B respectively. Therefore, var2 is visible only to window B, while var1 is visible to both window A and B.

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

[1] Built in id space owner includes UNIQ-javadoc-0-736dd3f9533a2ca2-QINU , UNIQ-javadoc-1-736dd3f9533a2ca2-QINU and macro components.

Declare a Local Variable

If a variable is declared inside a pair of curly braces, it is visible only to the scope defined by the curly braces. It is called a local variable. For example,

```
<zscript>
void echo() {
    String a_local_variable;
}
</script>
```

Here is another example,

```
<window>
    <zscript>
```

```
{  
    Date now = new Date(); //local variable  
    abc ="def"; //global variable since not defined before and  
not Class specified  
}  
String first = "first"; //global variable  
</zscript>  
0: ${first}  
1:${abc}  
2:${now}  
</window>
```

The result shows: 0: first 1: def 2: . It is because now is a local variable and it is invisible to EL expressions. On the other hand, first and abc are both global variables that are visible to EL expressions. Notice that abc is not declared but assigned directly, and it causes a global variable to be created.

Please refer to the Beanshell Documentation (<http://beanshell.org/docs.html>) and search "scoping" and "local" for more information.

Use Other Languages

Currently, zscript supports Java, Groovy, Ruby, JavaScript and Python. For example,

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

In addition, you could add your own interpreter by implementing Interpreter (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/scripting/Interpreter.html#>). For more information, please refer to ZUML Reference.

Conditional Evaluation

If and Unless

The component creation can be conditional. By specifying the `if`, `unless` attribute or both, developers can control whether to create the associated component. It is also the most straightforward way.

For example, suppose that we want to use label, if readonly, and textbox, otherwise:

```
<label value="${customer.label}" if="${param,readonly == 'true'}"/>
<textbox value="${customer.value}" unless="${param,readonly == 'true' }"/>
```

Besides, if a parent component is ignored (not created), all of its child components are ignored too.

Here is another example:

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

- window is created only if a is 1 and b is not 2.

Switch and Case

With the `switch` and `case` attributes of the `zk` element, you can evaluate a section of a ZUML document only if a variable matches a certain value. It is similar to Java's switch statement.

For example,

```
<zk switch="${fruit}">
  <zk case="apple">
    Evaluated only if ${fruit} is apple
  </zk>
  <zk case="${special}">
    Evaluated only if ${fruit} equals ${special}
  </zk>
  <zk>
    Evaluated only if none of the above cases matches.
  </zk>
</zk>
```

ZK Loader will evaluate from the first case to the last case, until it matches the switch condition, which is the value specified in the `switch` attribute. The evaluation is mutually exclusive conditional. Only the first matched case is evaluated.

The `zk` element without any case is the default – i.e., it always matches and is evaluated if all the cases above it failed to match.

Multiple Cases

You can specify a list of cases in one case attribute, such that a section of a ZUML document has to be evaluated if one of them matches.

```
<zk switch="${fruit}">
  <zk case="apple, ${special}">
    Evaluated if ${fruit} is either apple or ${special}
  </zk>
</zk>
```

Regular Expressions

Regular expressions are allowed in the case attribute too, as shown below.

```
<zk switch="${fruit}">
  <zk case="/ap*.e/">
    Evaluate if the regular expression, ap*.e., matches the switch
    condition.
  </zk>
</zk>
```

Used with forEach

Like any other elements, you can use the forEach attribute (so are if and unless). The forEach attribute is evaluated first, so the following is the same as multiple cases.

```
<zk case="${each}" forEach="apple, orange">
```

is equivalent to

```
<zk case= "apple, orange">
```

Choose and When

The choose and when attributes of the zk element are the third approach of conditional evaluation.

As shown below, it is enclosed with a zk element with the choose attribute, and the ZK Loader will evaluate its child elements (the zk elements with the when attribute) one-by-one until the first one matches:

```
<zk choose="">
  <zk when="${fruit == 'apple'}">
    Evaluated if the when condition is true.
  </zk>
  <zk><!-- default -->
    Evaluated if none of above cases matches.
  </zk>
</zk>
```

You don't have to assign any value to the choose attribute, which is used only to identify the range of the mutually exclusive conditional evaluation.

Iterative Evaluation

forEach

By default, ZK instantiates a component for each XML element. If you would like to generate a collection of components, you could specify the `forEach` attribute. For example,

```
<listbox>
  <listitem label="${each}" forEach="Apple, Orange, Strawberry"/>
</listbox>
```

is equivalent to

```
<listbox>
  <listitem label="Apple"/>
  <listitem label="Orange"/>
  <listitem label="Strawberry"/>
</listbox>
```

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

If you have a variable holding a collection of objects, you can specify it directly in the `forEach` attribute. For example, assume that you have a variable called `grades` as follows.

```
grades = new String[] {"Best", "Better", "Good"};
```

Then, you can iterate them by the use of the `forEach` attribute as follows. Notice that you have to use EL expression to specify the collection.

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

The iteration depends on the type of the value of the `forEach` attribute:

- If it is `java.util.Collection`, it iterates each element of the collection.
- if it is `java.util.Map`, it iterates each `Map.Entry` of the map.
- If it is `java.util.Iterator`, it iterates each element from the iterator.
- If it is `java.util.Enumeration`, it iterates each element from the enumeration.
- If it is `Object[]`, `int[]`, `short[]`, `byte[]`, `char[]`, `float[]` or `double[]`, it iterates each element from the array.
- If it is `null`, nothing is generated (it is ignored).
- If neither of the above types is specified, the associated element will be evaluated once as if a collection with a single item is specified.

The each Object

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

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

The forEachStatus Object

The `forEachStatus` object is an instance of `ForEachStatus`^[2]. It holds the information about the current iteration. It is mainly used to get the item of the enclosing element that is also assigned with the `forEach` attribute.

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

```
<hlayout>
    <zscript>
        classes = new String[] {"College", "Graduate"};
        grades = new Object[] {
            new String[] {"Best", "Better"}, new String[] {"A++", "A+", "A"}
        };
    </zscript>
    <listbox width="200px" forEach="#{classes}">
        <listhead>
            <listheader label="#{each}" />
        </listhead>
        <listitem label="#{forEachStatus.previous.each}: #{each}"
            forEach="#{grades[forEachStatus.index]}" />
    </listbox>
</hlayout>
```

Notice that the `each` and `forEachStatus` objects can be accessible both in an EL expression and in zscript.

Apply forEach to Multiple Elements

If you have to iterate a collection of items for multiple XML elements, you could group them with the `zk` element as shown below.

```
<zk forEach="#{cond}">
    ${each.name}
    <textbox value="#{each.value}" />
    <button label="Submit" />
</zk>
```

The `zk` element is a special element used to *group* a set of XML elements nested. ZK Loader will not create a component for it. Rather, it interprets the `forEach`, `if` and `unless` attribute it might have.

Access each and forEachStatus in Java

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

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

In a composer, you could retrieve them from the attributes, because these objects are actually stored in the parent component's attributes (`Component.getAttribute(java.lang.String)` [3]). For example,

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

If the component is a root, you could retrieve them from the page's attributes (`Page.getAttribute(java.lang.String)` [4]).

Access each and forEachStatus in Event Listeners

However, you cannot access the values of `each` and `forEachStatus` in an event listener because their values are reset after the XML element which `forEach` is associated has been evaluated.

For example, the following code will not work:

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

When the `onClick` event is received, the `each` object no longer exists.

There is a simple solution: store the value in the component's attribute, so you can retrieve it when the event listener is called. For example,

```
<button label="${each}" forEach="${countries}"
    onClick='alert(self.getAttribute("country"))'>
        <custom-attributes country="${each}" />
    </button>
```

Iterate a Subset of a Collection

If you would like to iterate a subset of a collection, you could specify the `forEachBegin` and/or `forEachEnd` attributes.

```
<grid>
  <rows>
    <row forEach="${foos}" forEachBegin="${param.begin}" forEachEnd="${param.end}">
      ${each.name} ${each.title}
    </row>
  </rows>
</grid>
```

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ForEachStatus.html#>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/ui/util/ForEachStatus.html#>
- [3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#getAttribute\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#getAttribute(java.lang.String))
- [4] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Page.html#getAttribute\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Page.html#getAttribute(java.lang.String))

On-demand Evaluation

By default, ZK creates components based on what is defined in a ZUML document when loading the document. However, we can defer the creation of some sections of components, until necessary, such as becoming visible. This technique is called load-on-demand or render-on-demand.

For example, you could split a ZUML document into multiple pages, and then load the required ones when necessary. Please refer to the Load ZUML in Java section for how to load a ZUML document dynamically.

It improves the performance both at the server and client sides. It is suggested to apply this technique whenever appropriate. In addition, ZK Loader provides a standard on-demand evaluation called *fulfill* to simplify the implementation as described in the following section.

Load-on-Demand with the `fulfill` Attribute

The simplest way to defer the creation of the child components is to use the `fulfill` attribute. For example, the `comboitem` in the following code snippet will not be created, until the `combobox` receives the `onOpen` event, indicating that `comboitem` is becoming visible.

```
<combobox fulfill="onOpen">
  <comboitem label="First Option"/>
</combobox>
```

In other words, if an XML element is specified with the `fulfill` attribute, all of its child elements will not be processed until the event specified as the value of the `fulfill` attribute is received.

Specify Target with ID or Implicit Variable

If the event to trigger the creation of children is targeted at another component, you can specify the target component's identifier in front of the event name as depicted below.

```
<button id="btn" label="show" onClick="content.visible = true"/>
<div id="content" fulfill="btn.onClick">
    Any content created automatically when btn is clicked
</div>
```

Create a Tabpanel's Children after It's Selected

```
<tabbox>
    <tabs>
        <tab selected="true">tab 1</tab>
        <tab>tab 2</tab>
    </tabs>
    <tabpanels>
        <tabpanel>
            ...
        </tabpanel>
        <tabpanel fulfill="self.linkedTab.onSelect">
            ...
        </tabpanel>
    </tabpanels>
</tabbox>
```

Specify Target with its Path

If the components belong to a different ID space, you can specify a path before the event name as follows:

```
<button id="btn" label="show" onClick="content.visible = true"/>
<window id="content" fulfill="../btn.onClick">
    Any content created automatically when btn is clicked
</window>
```

Specify Target with EL Expressions

EL expressions are allowed to specify the target, and it must return a component, an identifier or a path.

```
<button id="foo" label="click me to show"/>
<div fulfill="${foo}.onClick">
    created on demand
</div>
```

Specify Multiple Fulfill Conditions

If there are multiple conditions to fulfill, you could specify all of them in the fulfill attribute by separating them with a comma, such as

```
<div fulfill="b1.onClick, ${another}.onOpen">
...
</div>
```

Load Another ZUML on Demand with the fulfill Attribute

You could specify an URI in the fulfill attribute when the fulfill condition is satisfied (i.e. if a specified event has been received). The ZUML document of the URI will be loaded and rendered as the children of the associated component. To specify an URI, just append it to the condition and separate with an equal sign (=). For example,

```
<zk>
  <button id="btn" label="Click to Load"/>
  <div fulfill="btn.onClick=another.zul"/>
</zk>
```

Then, another.zul will be loaded when the button is clicked.

Notice that even though you could specify multiple conditions, you could specify at most one URI. The ZUML document of the URI will be loaded no matter which condition is satisfied.

```
<div fulfill="btn.onClick, foo.onOpen=another.zul"/>
```

If you specify an URI without any conditions, the ZUML document of the URI will be loaded from the very beginning. In other words, it has the same effect as using include.

```
<div fulfill="=another.zul"/>
```

The onFulfill Event

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

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

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

References

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

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

Include a Page

Apply (Recommended)

We suggest using `<apply>` ^[1] instead of `Include`. Because comparing to `Include`, it has several advantages:

1. Doesn't consume extra memory.

Because it's a shadow element, it doesn't create a corresponding component at the server side.

2. Doesn't render an extra `<div>` surrounding its child components at the client-side.

`Include` renders a `<div>` to enclose its child components. Sometimes the outer `<div>` breaks the layout.

3. Doesn't create an ID space.

`Include` itself is an ID space owner which affects you when locating a component with ZK selector syntax.

4. It can render its child components upon parameters dynamically.

You can pass a parameter with data binding and bind to the parameter. When the parameter's value changes, the content will change accordingly.

Include

`Include` allows you to include a `zul` page, an `HTML`, `zhtml`, `JSP` page, or a URL mapped to a servlet. For example,

```
<include src="another.zul"/>
<include src="another.jsp"/>
```

When including a ZUML page, the components specified in the ZUML page will become the child components of the `Include` ^[14] component.

For example, suppose we have two ZUL pages as follows:

```
<!-- first.zul -->
<include src="second.zul"/>
```

and

```
<!-- second.zul -->
<listbox>
  <listitem label="foo"/>
</listbox>
```

Then, `listbox` in `second.zul` will become the child of `include` in `first.zul`.

When including a non-ZUML page (such as JSP), the output of the page will be the content of the `Include` ^[14] component. Thus, the output must be a valid HTML fragment without `<html>`, `<head>`, `<body>`.

If you prefer to create an independent page (Page ^[8]), or want to include a page rendered by Richlet while the value of `src` ends with `.zul` or `.zhtml`, you could specify the mode with `defer` (`Include.setMode(java.lang.String)` ^[2]). Then, `include` won't have any child. Rather, an instance of Page ^[8] will be created to hold the content of `second.zul` or the content generated by Richlet. For more information, please refer to ZK Component Reference: `include`.

Classpath Web Resource Path

ZK provides a special path URL starting with `~./`, it looks for a file under a folder starting with `web` in a web application's classpath, e.g.

- `my-module.jar/web/`.
- `WEB-INF/classes/web`

So it will get a file under the path among all included jars. You can specify such URL in a component path-related attribute like:

```
<?component name="another" templateURI="~./another.zul" ?>
<zk>
    <vlayout>
        apply templateURI:
        <apply templateURI="~./another.zul" />

        component directive:
        <another/>

        image src:
        <image src="~./zklogo.png" />
    </vlayout>
</zk>
```

Modular Resource Sharing

It can be used as a default shared folder path for a jar. When you create a sub-module project, you can put some shared resources or reusable template zul files under this folder. Then, package the sub-module as a jar and include the sub-module jar in the main project. The main project can easily access those reusable resources by this special URL. Notice that the zul files under this special resource path are publicly accessed with a URL in a browser, you should not put any sensitive data in it.

Application-wide Named <Apply>

If you prefer an application-wide named `<apply>`^[3] element with a predefined templateURI and default parameters, you could specify it in a language addon. For example, we could prepare a file called `WEB-INF/lang-addon.xml` with the following content:

```
<language-addon>
    <addon-name>myapp</addon-name>
    <language-name>xul/html</language-name>
    <component>
        <component-name>mytemplatecomp</component-name>
        <template-uri>~./template/mytemplate.zul</template-uri>
    </component>
</language-addon>
```

Next, we could specify this file by adding the following content to `WEB-INF/zk.xml`:

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

```
</language-config>
```

Then, we can use it with a custom tag name:

```
<mytemplatecomp>
```

For more information, please refer to ZK Configuration Reference.

References

- [1] http://books.zkoss.org/zk-mvvm-book/8.0/shadow_elements/shadow_elements.html
- [2] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Include.html#setMode\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Include.html#setMode(java.lang.String))
- [3] <http://books.zkoss.org/zk-mvvm-book/8.0/syntax/apply.html>

Load ZUML in Java

Overview

Execution^[1] provides a collection of methods to allow you to create components based on a ZUML document, such as `org.zkoss.zk.ui.Component, java.util.Map` `Execution.createComponents(java.lang.String, org.zkoss.zk.ui.Component, java.util.Map)`^[2], `java.lang.String, org.zkoss.zk.ui.Component, java.util.Map` `Execution.createComponentsDirectly(java.lang.String, java.lang.String, org.zkoss.zk.ui.Component, java.util.Map)`^[3] and many others. In addition, Executions^[4] provides a similar collection of shortcuts so that you do not have to retrieve the current execution first.

For example,

```
public class Controller extends SelectorComposer {
    @Wire
    private Window main; //assumed wired automatically
    @Listen(onClick = "#main")
    public void createListbox() {
        Executions.createComponentsDirectly(
            "<listbox><listitem label=\"foo\"/></listbox>", "zul", this, null);
    }
    ...
}
```

Create from URI

There are several ways to create components based on a ZUML document. One of the most common approaches is to create components from a URI.

```
Map arg = new HashMap();
arg.put("myKey", someValue);
Executions.createComponents("/foo/my.zul", parent, arg); //attach to
page as root if parent is null
```

where `parent` (an instance of Component^[1]) will become the parent of the components specified in the ZUML document. If `parent` is null, the components specified in the ZUML documents will become the root components of the current page. In other words, the components created by `org.zkoss.zk.ui.Component, java.util.Map` `Execution.createComponents(java.lang.String, org.zkoss.zk.ui.Component, java.util.Map)`^[2] will be attached to the

current page.

The arg Object

The Map passed to the `createComponents()` can be accessed on the page being created by use of the `arg` object. For example,

```
<button label="Submit" if="${arg.myKey}" />
```

or

```
Executions.getCurrent().getArg().get("myKey");
```

Create Components Not Attached to Any Pages

If you want to create components that will not be attached to a page, you could use `java.util.Map` `Execution.createComponents(java.lang.String, java.util.Map)`^[2]. It is useful if you want to maintain a cache of components or implement a utility.

For example:

```
Map arg = new HashMap();
arg.put("someName", someValue);
Component[] comps =
Executions.getCurrent().createComponents("/foo/my.zul", arg); //won't
be attached to a page
cache.put("pool", comps); //you can store and use them later since they
are not (yet) attached to any pages
```

Create Components in Working Thread

With `java.lang.String, java.util.Map` `Executions.createComponents(org.zkoss.zk.ui.WebApp, java.lang.String, java.util.Map)`^[5], you could create components in a working thread without execution^[6], though it is rare.

Of course, the components being created by `java.lang.String, java.util.Map` `Executions.createComponents(org.zkoss.zk.ui.WebApp, java.lang.String, java.util.Map)`^[5] will not be attached to any pages. You have to attach them manually, if you want to show them to the client.

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

[2] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#createComponents\(java.lang.String,java.util.Map\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#createComponents(java.lang.String,java.util.Map))

[3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#createComponentsDirectly\(java.lang.String,java.util.Map\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#createComponentsDirectly(java.lang.String,java.util.Map))

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

[5] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#createComponents\(org.zkoss.zk.ui.WebApp,java.lang.String,java.util.Map\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#createComponents(org.zkoss.zk.ui.WebApp,java.lang.String,java.util.Map))

[6] It means `Execution` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#>) returns null. For example, it happens when the application starts, or in a working thread.

Create from Content Directly

If the ZUML document is a resource of Web application (i.e., not accessible through `ServletContext`), you could use one of the `createComponentsDirectly` methods. For example, you could read the content into a string from database and pass it to `java.lang.String, org.zkoss.zk.ui.Component, java.util.Map` `Execution.createComponentsDirectly(java.lang.String, java.lang.String, org.zkoss.zk.ui.Component, java.util.Map)` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#createComponentsDirectly\(java.lang.String,java.lang.String,org.zkoss.zk.ui.Component,java.util.Map\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#createComponentsDirectly(java.lang.String,java.lang.String,org.zkoss.zk.ui.Component,java.util.Map))). Or, you could represent the content as a reader (say, representing BLOB in database) and then pass it

to `java.lang.String, org.zkoss.zk.ui.Component, java.util.Map`) `Execution.createComponentsDirectly(java.io.Reader, java.lang.String, org.zkoss.zk.ui.Component, java.util.Map)` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#createComponentsDirectly\(java.io.Reader,%20java.lang.String,%20org.zkoss.zk.ui.Component,%20java.util.Map\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#createComponentsDirectly(java.io.Reader,%20java.lang.String,%20org.zkoss.zk.ui.Component,%20java.util.Map)))

For example, suppose we want to create a component from a remote site. Then, we could represent the resource as a URL and do as follows.

```
public void loadFromWeb(java.net.URL src, Component parent) {
    Executions.createComponentsDirectly(
        new java.io.InputStreamReader(src.openStream(), "UTF-8"),
        parent, null);
}
```

Create from Page Definition

When creating components from the URI (such as `org.zkoss.zk.ui.Component, java.util.Map`) `Execution.createComponents(java.lang.String, org.zkoss.zk.ui.Component, java.util.Map)` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#createComponents\(java.lang.String,%20org.zkoss.zk.ui.Component,%20java.util.Map\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#createComponents(java.lang.String,%20org.zkoss.zk.ui.Component,%20java.util.Map))), ZK Loader will cache the parsed result and reuse it to speed up the rendering.

However, if you create components from the content directly (such as `java.lang.String, org.zkoss.zk.ui.Component, java.util.Map`) `Execution.createComponentsDirectly(java.lang.String, java.lang.String, org.zkoss.zk.ui.Component, java.util.Map)` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#createComponentsDirectly\(java.lang.String,%20java.lang.String,%20org.zkoss.zk.ui.Component,%20java.util.Map\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#createComponentsDirectly(java.lang.String,%20java.lang.String,%20org.zkoss.zk.ui.Component,%20java.util.Map))), there is no way to cache the parsed result. In other words, the ZUML content will be parsed each time `createComponentsDirectly` is called.

It is OK if the invocation does not happen frequently. However, if you want to improve the performance, you could parse the content into `PageDefinition` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/metainfo/PageDefinition.html#>) by using `java.lang.String, java.lang.String` `Executions.getPageDefinitionDirectly(org.zkoss.zk.ui.WebApp, java.lang.String, java.lang.String)` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#getPageDefinitionDirectly\(org.zkoss.zk.ui.WebApp,%20java.lang.String,%20java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#getPageDefinitionDirectly(org.zkoss.zk.ui.WebApp,%20java.lang.String,%20java.lang.String))), cache it, and then invoke `org.zkoss.zk.ui.Component, java.util.Map` `Executions.createComponents(org.zkoss.zk.ui.metainfo.PageDefinition, org.zkoss.zk.ui.Component, java.util.Map)` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#createComponents\(org.zkoss.zk.ui.metainfo.PageDefinition,%20org.zkoss.zk.ui.Component,%20java.util.Map\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#createComponents(org.zkoss.zk.ui.metainfo.PageDefinition,%20org.zkoss.zk.ui.Component,%20java.util.Map))) to create them repeatedly.

`PageDefinition` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/metainfo/PageDefinition.html#>) is a Java object representing a ZUML document. It is designed to allow ZK Loader to interpret even more efficiently. Unfortunately, it is not serializable, so you can not store it into database or other persistent storage. You could serialize or marshal the original content (i.e., ZUML document) if required.

Notices

There are a few notices worth to know.

No Page Created

When creating components from a ZUML document as described above, no page (`Page` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Page.html#>)) is created. Components are attached to the current page, to a component, or simply standalone. Since no page is created, there are a few differences than visiting a ZUML document directly^[1].

1. The <?page?>, <?script?>, <?link?>, <?header?> and other directives controlling a page (Page (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Page.html#>)) have no function. It means that you could not change the page's title, add JavaScript code, or add CSS with these directives in a ZUML document loaded in this way.
2. On the other hand, when <?function-mapper?>, <?variable-resolver?> and <?component?> work correctly, they decide how a ZUML document is parsed rather than how the current page (Page (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Page.html#>)) should be.
3. The variables, functions and classes defined in zscript will be stored in the interpreter of the current page (Page.getInterpreter(java.lang.String) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Page.html#getInterpreter\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Page.html#getInterpreter(java.lang.String)))).
 - If java.util.Map) Execution.createComponents(java.lang.String, java.util.Map) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#createComponents\(java.lang.String,java.util.Map\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#createComponents(java.lang.String,java.util.Map))), java.lang.String, java.util.Map) Executions.createComponents(org.zkoss.zk.ui.WebApp, java.lang.String, java.util.Map) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#createComponents\(org.zkoss.zk.ui.WebApp,java.lang.String,java.util.Map\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#createComponents(org.zkoss.zk.ui.WebApp,java.lang.String,java.util.Map))) or similar is used to create components not attached to any page, the variables, functions and classes defined in the ZUML document will be lost. Thus, it is *not* a good idea to use zscript in this case.

[1] Don't confuse a ZUML page with org.zkoss.zk.ui.Component, java.util.Map) Execution.createComponents(java.lang.String, org.zkoss.zk.ui.Component, java.util.Map) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#createComponents\(java.lang.String,org.zkoss.zk.ui.Component,java.util.Map\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#createComponents(java.lang.String,org.zkoss.zk.ui.Component,java.util.Map))). The former refers to a file containing a ZUML document. The latter is a Java object of java.lang.String, org.zkoss.zk.ui.Component, java.util.Map) Execution.createComponentsDirectly(java.lang.String, java.lang.String, org.zkoss.zk.ui.Component, java.util.Map) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#createComponentsDirectly\(java.lang.String,java.lang.String,org.zkoss.zk.ui.Component,java.util.Map\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#createComponentsDirectly(java.lang.String,java.lang.String,org.zkoss.zk.ui.Component,java.util.Map))) that represents a portion of a desktop.

XML Namespaces

In a ZUML document, an XML namespace is used to identify either a special functionality or a component set. We call the former a standard namespace and the latter a language.

Standard Namespaces

For example, the client namespace is used to indicate that an XML attribute shall be interpreted as a client-side control.

In the following example, w:onFocus is a client-side listener since w: is specified, while onChange is an event attribute of a component.

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

The native namespace is another standard namespace used to indicate that an XML element should be generated *natively* rather than a component. For example,

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

```
</n:tr>
</n:table>
```

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

For more information, please refer to ZUML Reference.

Languages

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

For example, Window^[1], Button^[2] and Combobox^[2] all belong to the same language called xul/html. It is a ZK variant of XUL (and also known as zul).

Component designers are free to designate a component definition to any component sets they prefer, as long as there is no name conflict.

When parsing a ZUML document, ZK Loader has to decide the language that an XML element is associated, so that the correct component definition (ComponentDefinition^[3]) can be resolved. For example, in the following example, ZK needs to know window belongs to the xul/html language, so its component definition can be retrieved correctly.

```
<window>
```

ZK Loader first decides the default language from the extension. For example, foo.zul implies the default language is ZUL. The default language is used if an XML element is not specified with any XML namespace. For example, window in the previous example will be considered as a component definition of the ZUL language.

If the extension is zhtml (such as foo.zhtml), the default language will be XHTML. Thus, window in the previous example will be interpreted incorrectly. To solve it, you could specify the XML namespace explicitly as follows.

```
<!-- foo.zhtml -->
<p> <!-- assumed from the XHTML language -->
  <u:window xmlns:u="zul"/> <!-- ZK Loader will search the ZUL language instead -->
</p>
```

For more information about identifying a language, please refer to ZUML Reference.

Version History

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

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/metainfo/LanguageDefinition.html#>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Combobox.html#>
- [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/metainfo/ComponentDefinition.html#>

Richlet

Overview

A richlet is a small Java program that composes a user interface in Java for serving the user's request. Before composing UI in Java, we suggest you to know basic concept: UI Composing/Component-based UI first.

When a user requests the content of an URL, ZK Loader checks if the resource of the specified URL is a ZUML page or a richlet. If it is a ZUML page, ZK Loader will create components automatically based on the ZUML page's content as we described in the previous chapters.

If the resource is a richlet, ZK Loader hands over the processing to the richlet. What and how to create components are all handled by the richlet. In other words, it is the developer's job to create all the necessary components programmatically in response to the request.

The choice between the ZUML pages and richlets depends on your preference. However, the performance should not cause any concern since parsing ZUML is optimized.

Implement a Richlet

It is straightforward to implement a richlet. First, you have to implement the Richlet^[1] interface before mapping a URL to the richlet.

Implement a Richlet as a Java class

A richlet must implement the Richlet^[1] interface. However, you generally do not have to implement it from scratch. Rather, you could extend GenericRichlet^[2], and the only thing you have to do is to override Richlet.service(org.zkoss.zk.ui.Page)^[13]. The method is called when an associated URL is requested. For example,

```
package org.zkoss.reference.developer.uicomposing;

import org.zkoss.zk.ui.*;
import org.zkoss.zk.ui.event.*;
import org.zkoss.zul.*;

public class TestRichlet extends GenericRichlet {
    //Richlet//
    public void service(Page page) {
        page.setTitle("Richlet Test");

        final Window w = new Window("Richlet Test", "normal",
false);
        new Label("Hello World!").setParent(w);
        final Label l = new Label();
        l.setParent(w);

        final Button b = new Button("Change");
        b.addEventListener(Events.ON_CLICK,
            new EventListener() {
                int count;
```

```

        public void onEvent(Event evt) {
            l.setValue(" " + ++count);
        }
    );
    b.setParent(w);

    w.setPage(page);
}

@Override
public void init(RichletConfig config) {
    super.init(config);
    //initialize resources
}

@Override
public void destroy() {
    super.destroy();
    //destroy resources
}
}
}

```

In Richlet, you have to compose UI on your own, but some components only support specific child components. We recommend you to read ZK Component Reference before you start to build.

As shown above (line 27), we have to invoke page) Component setPage(Page page) ^[3] explicitly to attach a root component to a page so it will be available at the client.

To have better control, you can even implement the Richlet.init(org.zkoss.zk.ui.RichletConfig) ^[4] and Richlet.destroy() ^[5] methods to initialize and to destroy any resources required by the richlet when it is loaded.

In addition, you could implement Richlet.getLanguageDefinition() ^[6] to use a different language as default (for example, implementing a richlet for mobile devices ^[7]). By default, ZUL (aka., xul/html) is assumed.

Richlet Must Be Thread-Safe

Like a servlet, a single instance of richlet is created and shared with all users for all requests for the mapped URL. A richlet must handle the concurrent requests, and be careful to synchronize access to shared resources. In other words, a richlet (the implementation of the service method) must be thread-safe.

Don't Share Components

When a request (not Ajax request but regular HTTP request) is made by a user, a Desktop ^[9] and a Page ^[8] are created first, and then Richlet.service(org.zkoss.zk.ui.Page) ^[13] is invoked to serve the request^[8]. In other words, each request is served with an individual desktop and page. Therefore, we *cannot* share components among different invocations of Richlet.service(org.zkoss.zk.ui.Page) ^[13].

For example, the following code is illegal:

```

public class MyRichlet extends GenericRichlet {
    private Window main; //Not a good idea to share
    public void service(Page page) {
}
}

```

```

if (main == null) {
    main = new Window();
}
main.setPage(main); //ERROR! Causes an exception if the same
URL is requested twice!
...

```

Why? Each desktop should have its own set of component instances^[9]. When the URL associated **MyRichlet** is requested a second time, an exception will be thrown because the **main** window is already instantiated and associated with the first desktop created from the first request. We cannot assign it to the second desktop.

-
- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Richlet.html#>
 - [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/GenericRichlet.html#>
 - [3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#setPage\(Page\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#setPage(Page))
 - [4] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Richlet.html#init\(org.zkoss.zk.ui.RichletConfig\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Richlet.html#init(org.zkoss.zk.ui.RichletConfig))
 - [5] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Richlet.html#destroy\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Richlet.html#destroy())
 - [6] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Richlet.html#getLanguageDefinition\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Richlet.html#getLanguageDefinition())
 - [7] <http://code.google.com/p/zkreach/>
 - [8] A normal HTTP request; not an Ajax request. Ajax requests are handled in the same way as ZUML. For more information please refer to the Event Handling section
 - [9] For more information, please refer to Component-based UI section

Map URL to a Richlet

To map URL to a richlet, there are two steps.

1. Turn on the support of Richlet (in WEB-INF/web.xml)
2. Map URL pattern to Richlet (in WEB-INF/zk.xml)

Turn on Richlet

By default, richlets are disabled. To enable them, please add the following declaration to WEB-INF/web.xml. Once enabled, you can add as many richlets as you want without modifying web.xml.

With servlet-mapping:

```

<servlet-mapping>
    <servlet-name>zklLoader</servlet-name>
    <url-pattern>/zk/*</url-pattern>
</servlet-mapping>

```

You can use RichletFilter instead.

```

<filter>
    <filter-name>RichletFilter</filter-name>
    <filter-class>org.zkoss.zk.ui.http.RichletFilter</filter-class>
</filter>

<filter-mapping>
    <filter-name>RichletFilter</filter-name>
    <url-pattern>/zk/*</url-pattern>
</filter-mapping>

```

where you can replace `/zk/*` with any pattern you like, such as `/do/*`. Notice that you *cannot* map it to an extension (such as `*.do`) since it will be considered as a ZUML page (rather than a richlet).

Map URL pattern to Richlet

For each richlet you implement, you can define it in `WEB-INF/zk.xml` with the statement similar to the following:

```
<richlet>
    <richlet-name>Test</richlet-name><!-- your preferred name -->
    <richlet-class>org.zkoss.zkdemo.TestRichlet</richlet-class><!-- your class name, of course -->
</richlet>
```

After defining a richlet, you can map it to any number of URLs using the `richlet-mapping` element as shown below.

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

Note: With Richlet Filter (since ZK 7.0.0), you should add the prefix of url-pattern of the filter-mapping into the url-pattern of richlet-mapping. For example,

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

As you can see in the highlight above, the `/zk` is added which is according to the filter-mapping.

Then, you can visit `http://localhost:8080/PROJECT_NAME/zk/test` (`http://localhost:8080/PROJECT_NAME/zk/test`) to request the richlet.

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

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

Tip: By specifying `/*` as the `url-pattern`, you can map all unmatched URLs to your richlet.

Load ZUML in Richlet

Execution (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#>) provides a collection of methods, such as `org.zkoss.zk.ui.Component`, `java.util.Map`) `Execution.createComponents(java.lang.String, org.zkoss.zk.ui.Component, java.util.Map)` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#createComponents\(java.lang.String,java.util.Map\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#createComponents(java.lang.String,java.util.Map))), allowing developers to load ZUML documents dynamically. You could load a ZUML document from any source you like, such as database. Please refer to the Load ZUML in Java for details.

Use Spring in Richlet

To use Spring-managed beans in richlets you need the context loader listener that creates spring application context as described in [ZK Spring Essentials/Getting Started with ZK Spring/Setting Up ZK Spring](#). Then you could load Spring beans by using a utility class `SpringUtil` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkplus/spring/SpringUtil.html#>):

```
Object bean = SpringUtil.getBean(beanName);
```

Macro Component

There are two ways to implement a component. One is to implement a component in a Java class, extending from other component or one of the skeletal implementations with an optional JavaScript class. It is flexible and, technically, is also able to implement any functionality you want. For more information please refer to [ZK Component Development Essentials](#).

On the other hand, we could implement a new component by using the others and composing them in a ZUML page. In other words, we could define a new component by expressing it in a ZUML page. It works like composition, macro expansion, or inline replacement.

For the sake of convenience, we call the first type of components *primitive components* and the second type *macro components*. In this section we will get into more details on how to implement a macro component and how to use it.

There is a similar concept called composite components. Unlike macros, you could derive from any component but you have to do the loading of ZUML manually. For more information please refer to the Composite Component section.

Definition, Declaration and Use

It is straightforward to apply macro components to an application:

1. Define (aka., Implement) a macro component in a ZUML page.
2. Declare the macro component in the page or the whole application that is going to use the macro component.
3. Use the macro components. The use of a macro component is the same as using primitive components.

Define Macro Component

The definition of a macro component is expressed in a ZUML page. In other words, the page is the template of the macro component. It is the same as any other ZUML pages as it does not require any special syntaxes at all. Furthermore, any ZUML page can be used as a macro component too.

For example, assume that we want to pack a label and a text box as a macro component. Then we could create a page, say /WEB-INF/macros/username.zul, as follows.

```
<hlayout>
    Username: <textbox/>
</hlayout>
```

Declare Macro Component

Before using a macro component, you have to declare it first. It is straightforward to use component directives. For example, we could add the first line to the page that is going to use the *username* macro component:

```
<?component name="username" macroURI="/WEB-INF/macros/username.zul"?>
```

As shown, we have to declare the component's name (the `name` attribute) and the URI of the page defining the macro component (the `macroURI` attribute).

If you prefer to make a macro component available to all pages, you could add the component definition to the so-called language addon and add it to WEB-INF/zk.xml.

Use Macro Component

Using a macro component in a ZUML page is the same as the use of any other components. There is no difference at all.

```
<window>
    <username/>
</window>
```

Pass Properties to Macro Component

Like an ordinary component, you can specify properties (a.k.a., attributes) when using a macro component. For example,

```
<?component name="username" macroURI="/WEB-INF/macros/username.zul"?>
<window>
    <username who="John" label="Username"/>
</window>
```

All these properties specified are stored in a map that is then passed to the template (aka., the macro definition; `macroURI`) via a variable called `arg`. Then, from the template, you could access these properties by the use of EL

expressions as shown below:

```
<hlayout>
    ${arg.label}: <textbox value="${arg.who}" />
</hlayout>
```

arg.includer

In addition to properties (aka., attributes), a property called `arg.includer` is always passed. It refers to the macro component itself. With this, we could reference other information such as its parent:

```
${arg.includer.parent}
```

Notice that `arg.includer` is different from the so-called inline macros. The inline macros are special macro components and used for inline expansion. For more information please refer to [Inline Macros](#) section.

Pass Initial Properties

Sometimes it is helpful to pass a list of initial properties that will be used to initialize a component when it is instantiated. It can be done easily as follows.

```
<?component name="mycomp" macroURI="/macros/mycomp.zul"
    myprop="myval" another="anotherval"?>
```

Therefore,

```
<mycomp/>
```

is equivalent to

```
<mycomp myprop="myval1" another="anotherval"/>
```

Control Macro in Java

Instantiate Macro in Java

To instantiate a macro component in Java, you could do the followings.

1. Looks up the component definition (`ComponentDefinition`^[3]) with the use of `boolean` `Page.getComponentDefinition(java.lang.String, boolean)`^[1].
2. Invokes `java.lang.String) ComponentDefinition.newInstance(org.zkoss.zk.ui.Page, java.lang.String)`^[2] to instantiate the component.
3. Invokes `Component.setParent(org.zkoss.zk.ui.Component)`^[3] to attach the macro to a parent, if necessary.
4. Invokes `Component.applyProperties()`^[4] to apply the initial properties defined in the component definition.
5. Invokes `java.lang.Object) DynamicPropertyed.setDynamicProperty(java.lang.String, java.lang.Object)`^[5] to assign any properties you want.
6. Finally, invokes `AfterCompose.afterCompose()`^[6] to create components defined in the template

For example,

```
HtmlMacroComponent ua = (HtmlMacroComponent)
    page.getComponentDefinition("username", false).newInstance(page,
null);
ua.setParent(wnd);
```

```
ua.applyProperties(); //apply properties defined in the component  
definition  
ua.setDynamicProperty("who", "Joe");  
ua.afterCompose(); //then the ZUML page is loaded and child components  
are created
```

It is a bit tedious. If you implement your own custom Java class (instead of `HtmlMacroComponent` [7]), it will be simpler. For example,

```
Username ua = new Username();  
ua.setParent(wnd);  
ua.setWho("Joe");
```

Please refer to the Implement Custom Java Class section for details.

Change Template at Runtime

You could change the template dynamically by the use of `HtmlMacroComponent.setMacroURI(java.lang.String)` [8]. For example,

```
<username id="ua"/>  
<button onClick="ua.setMacroURI("another.zul")">
```

If the macro component was instantiated, all of its children will be removed first, and then the new template will be applied (so-called recreation).

References

- [1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Page.html#getComponentDefinition\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Page.html#getComponentDefinition(java.lang.String))
- [2] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/metainfo/ComponentDefinition.html#newInstance\(org.zkoss.zk.ui.Page\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/metainfo/ComponentDefinition.html#newInstance(org.zkoss.zk.ui.Page))
- [3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#setParent\(org.zkoss.zk.ui.Component\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#setParent(org.zkoss.zk.ui.Component))
- [4] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#applyProperties\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#applyProperties())
- [5] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/ext/DynamicPropertyed.html#setDynamicProperty\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/ext/DynamicPropertyed.html#setDynamicProperty(java.lang.String))
- [6] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/ext/AfterCompose.html#afterCompose\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/ext/AfterCompose.html#afterCompose())
- [7] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlMacroComponent.html#>
- [8] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlMacroComponent.html#setMacroURI\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlMacroComponent.html#setMacroURI(java.lang.String))

Inline Macros

Overview

An inline macro is a special macro component which behaves like *inline-expansion*. Unlike a regular macro component, ZK does not create a macro component. Rather, it inline-expands the components defined in the macro URI, as if the content of the in-line macro's template is entered directly into the target page.

Declare an Inline Macro

To declare an inline macro, we have to specify `inline="true"` in the component directive, while the definition and the use of an inline macro are the same as the regular macro components (i.e., non-inline).

For example, suppose we have a macro definition (aka., template) as follows:

```
<!-- username.zul: (macro definition) -->
<row>
    Username
    <textbox id="${arg.id}" value="${arg.name}" />
</row>
```

We can declare it as an inline macro as follows:

```
<!-- target page -->
<?component name="username" inline="true" macroURI="username.zul"?>
<grid>
    <rows>
        <username id="ua" name="John"/>
    </rows>
</grid>
```

Then, it is equivalent to:

```
<grid>
    <rows>
        <row>
            Username
            <textbox id="ua" value="John"/>
        </row>
    </rows>
</grid>
```

Notice that all the properties, including `id`, are passed to the inline macro too.

Inline versus Regular Macro

As described above, an inline macro is expanded inline when it is used as if they are entered directly. On the other hand, ZK will create a component (an instance of `HtmlMacroComponent`^[7] or deriving) to represent the regular macro. That is, the macro component is created as the parent of the components that are defined in the template.

Inline macros are easier to integrate into sophisticated pages. For example, you *cannot* use *regular* macro components in the previous example since `rows` accepts only `row` as children, not macro components. It is also easier to access all components defined in a macro since they are expanded inline. However, it also means that the developers must take care of `id` themselves. In addition, there is no way to instantiate inline macros in pure Java (rather, `org.zkoss.zk.ui.Component`, `java.util.Map`) `Execution.createComponents(java.lang.String, org.zkoss.zk.ui.Component, java.util.Map)`^[2] shall be used)^[1].

On the other hand, regular macros allow the component developers to provide a custom Java class to represent the component so a better abstraction and addition functionality can be implemented. We will discuss it more in the following section.

[1] ZK Loader does create a component for an inline macro when rendering, and then drop it after *expanding* into the parent component.

Technically, an application can do the same thing but it is not recommended since we might change it in the future.

arg.includer

Unlike regular macros, `arg.includer` for an inline macro is the parent component of the macro (after all, the inline macro does not really exist).

An Example

inline.zul: (the macro definition)

```
<row>
    <textbox value="${arg.col1}" />
    <textbox value="${arg.col2}" />
</row>
```

useinline.zul: (the target page)

```
<?component name="myrow" macroURI="inline.zul" inline="true"?>
<window title="Test of inline macros" border="normal">
    <zscript><![CDATA[
        import org.zkoss.util.Pair;
        List infos = new LinkedList();
        for(int j = 0; j <10; ++j){
            infos.add(new Pair("A" + j, "B" + j));
        }
    ]]>
    </zscript>
    <grid>
        <rows>
            <myrow col1="${each.x}" col2="${each.y}" forEach="${infos}" />
        </rows>
    </grid>
</window>
```

Implement Custom Java Class

Overview

As described in the earlier sections, a macro component is instantiated to represent a regular macro. By default, `HtmlMacroComponent`^[7] is assumed (and instantiated). However, you can also provide a custom Java class to provide a better API to simplify the access and to encapsulate the implementation.

Implement Custom Java Class for Macro

The implementation is straightforward. First, the custom Java class for macro components must extend from `HtmlMacroComponent`^[7]. Second, though optional, it is suggested to invoke `HtmlMacroComponent.compose()`^[1] in the constructor^{[2][3]}, such that the template and the wiring of the data members will be applied in the constructor.

For example, suppose we have a macro template as follows.

```
<hlayout id="mc_layout">
    Username: <textbox id="mc_who"/>
</hlayout>
```

Then, we could implement a Java class for it:

```
package foo;

import org.zkoss.zk.ui.select.annotation.*;
import org.zkoss.zk.ui.HtmlMacroComponent;
import org.zkoss.zul.Textbox;

@VariableResolver(org.zkoss.zkplus.spring.DelegatingVariableResolver)
public class Username extends HtmlMacroComponent {
    @WireVariable
    private User currentUser; //will be wired if currentUser is a
Spring-managed bean, when compose() is called
    @Wire
    private Textbox mc_who; //will be wired when compose() is called
    public Username() {
        compose(); //for the template to be applied, and to wire
members automatically
    }
    public String getWho() {
        return mc_who.getValue();
    }
    public void setWho(String who) {
        mc_who.setValue(who);
    }
    @Listen("onClick=#submit")
    public void submit() { //will be wired when compose() is called.
    }
}
```

As shown, `HtmlMacroComponent.compose()`^[1] will wire variables, components and event listeners automatically, so we could access them directly (such as the `mc_who` member). For more information, please refer to the Wire Components section, the Wire Variables section and Wire Event Listeners sections.

Also notice that the `arg` variable is still available to the template so as to represent properties set by `java.lang.Object`) `DynamicPropertyed.setDynamicProperty(java.lang.String, java.lang.Object)`^[5], though it is more useful if a custom implementation is provided.

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlMacroComponent.html#compose\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlMacroComponent.html#compose())

[2] By default, `HtmlMacroComponent` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlMacroComponent.html#>) is invoked when is called. In many cases, it is generally too late, so we suggest to invoke it in the constructor.

[3] `HtmlMacroComponent.compose()` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlMacroComponent.html#compose\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlMacroComponent.html#compose())) is introduced in 5.0.5. For 5.0.4 or earlier, please invoke UNIQ-javadoc-3-736dd3f9533a2ca2-QINU instead.

Declare Macro with Custom Java Class

To make ZK Loader know which custom Java class to use, we have to specify the `class` attribute when declaring it in the component directives. For example,

```
<?component name="username" macroURI="/WEB-INF/macros/username.zul"
    class="foo.Username"?>
```

Use Macro with Custom Java Class

In ZUML

The use of the macro component with a custom Java class in a ZUML page is the same as other macro components.

In Java

The main purpose of introducing a custom Java class is to simplify the use of a macro component in Java. For example, you could invoke a more meaningful setter, say, `setWho`, directly rather than `java.lang.Object`) `DynamicPropertyed.setDynamicProperty(java.lang.String, java.lang.Object)` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/ext/DynamicPropertyed.html#setDynamicProperty\(java.lang.String,java.lang.Object\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/ext/DynamicPropertyed.html#setDynamicProperty(java.lang.String,java.lang.Object))). In addition, the instantiation could be as simple as follows:

```
Username ua = new Username();
ua.setParent(wnd);
ua.setWho("Joe");
```

Macro Component and ID Space

Like `Window` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Window.html#>), `HtmlMacroComponent` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlMacroComponent.html#>) also implements `IdSpace` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/IdSpace.html#>). It means that a macro component (excluding inline macros) is a space owner. In other words, it is free to use whatever identifiers to identify components inside the template.

For example, assume we have a macro defined as follows.

```
<hlayout>
    Username: <textbox id="who" value="${arg.who}" />
</hlayout>
```

Then, the following codes work correctly.

```
<?component name="username" macroURI="/WEB-INF/macros/username.zul"?>
<zkc>
    <username/>
    <button id="who"/> <!-- no conflict because it is in a different ID space -->
</zkc>
```

However, the following codes *do not* work.

```
<?component name="username" macroURI="/WEB-INF/macros/username.zul"?>
<username id="who"/>
```

Why? Like any ID space owner, the macro component itself is in the same ID space as its child components. There are two alternative solutions:

1. Use a special prefix for the identifiers of child components of a macro component. For example, "mc_who" instead of "who".

```
<hlayout>
    Username: <textbox id="mc_who" value="${arg.who}" />
</hlayout>
```

2. Use the `window` component to create an additional ID space.

```
<window>
    <hlayout>
        Username: <textbox id="who" value="${arg.who}" />
    </hlayout>
</window>
```

The first solution is suggested, if applicable, due to the simplicity.

Manipulate component inside macro component

As the code described above, the component is wired and composed in a constructor. Thus, you can append wired components or remove wired components in a `setProperty` method.

For example,

```
package foo;

import org.zkoss.zk.ui.select.annotation.*;
import org.zkoss.zk.ui.HtmlMacroComponent;
import org.zkoss.zul.Textbox;

@VariableResolver(org.zkoss.zkplus.spring.DelegatingVariableResolver)
public class Username extends HtmlMacroComponent {
    @WireVariable
    private User currentUser; //will be wired if currentUser is a
    Spring-managed bean, when compose() is called

    //Wire existing components
    @Wire
```

```

private Textbox mc_who;
@Wire
private Hlayout mc_layout;
public Username() {
    compose(); //for the template to be applied, and to wire
members automatically
}
public String getGender() {
    return currentUser.getGender();
}
public void setGender(String gender) {
    // append another textbox to hlayout
    Textbox genderTbx = new Textbox();
    genderTbx.setValue(gender);
    genderTbx.setParent(mc_layout);
}
}

```

Also, you can add a forward event to the newly added component and forward the event to a macro component.

```

public class Username extends HtmlMacroComponent {
    // omitted

    @Wire
    private Hlayout mc_layout;
    public void setGender(String gender) {
        Textbox genderTbx = new Textbox();
        genderTbx.setValue(gender);
        genderTbx.setParent(mc_layout);
        // listen onChange event to the textbox and forward to macro
component
        genderTbx.addForward(Events.ON_CHANGE, this, "onGenderChange",
        genderTbx.getValue());
    }
}

```

Then use the forward event to communicate with other components.

```

<?component name="username" macroURI="/WEB-INF/macros/username.zul" class="foo.Username"?>
<window apply="org.zkoss.bind.BindComposer" viewModel="@id('vm') @init('foo.MacroVM')">
    <username who="John" label="Username" gender="@load(vm.gender)" onGenderChange="@command('changeGender')"/>
</window>

```

Version History

Version	Date	Content
5.0.5	October, 2010	HtmlMacroComponent.compose() (http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlMacroComponent.html#compose()) was introduced.

Composite Component

Like a macro component, a composite component is an approach to compose a component based on a template. Unlike a macro component, a composite component has to create and wire the child components by itself, and handle ID space if necessary. The advantage is that a composite component can extend from any component, such as Row^[1], such that it is easier to fit to any situation (and no need for the inline concept).

In short, it is suggested to use a macro component if applicable (since it is easier), while using a composite component otherwise.

If you'd like to assemble UI at runtime (aka., templating), please refer to the Templating section for more information.

Implement a Composite Component

First, you have to decide which component to extend from. Div^[2] is a common choice as it is a simple component. However, here our example extends from Row^[1], so it can be used under Rows^[5], which the regular macros cannot.

Second, you have to implement a template (in a ZUML document) to define what child components the composite component has. Then, you have to implement a Java class to put them together.

Implement a Template

The implementation of a template is straightforward. There is nothing special to handle. Since it is rendered by org.zkoss.zk.ui.Component, java.util.Map Execution.createComponents(java.lang.String, org.zkoss.zk.ui.Component, java.util.Map)^[2], you could pass whatever data you prefer to it (through the arg argument).

Suppose we have a template as follows, and it is placed at /WEB-INF/composite/username.zul.

```
<zk>
  Username: <textbox id="mc_who"/>
</zk>
```

Implement a Java Class

To implement a Java class we shall:

1. Extend from the component class you want.
2. (Optional) Implement IdSpace^[1] to make it an ID space owner.
3. Render the template in the constructor by the use of org.zkoss.zk.ui.Component, java.util.Map) Executions.createComponents(java.lang.String, org.zkoss.zk.ui.Component, java.util.Map)^[3] or others.
4. (Optional) Wire variables, components and event listeners after rendering with the use of java.lang.Object, java.util.List) Selectors.wireVariables(org.zkoss.zk.ui.Component, java.lang.Object, java.util.List)^[4] (wiring variables), java.lang.Object, boolean) Selectors.wireComponents(org.zkoss.zk.ui.Component, java.lang.Object, boolean)^[5] (wiring components) and java.lang.Object) Selectors.wireEventListeners(org.zkoss.zk.ui.Component, java.lang.Object)^[6] (wiring event listeners).

For example,

```
package foo;

import org.zkoss.zk.ui.IdSpace;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zul.Row;
import org.zkoss.zul.Textbox;

public class Username extends Row implements IdSpace {
    @Wire
    private Textbox mc_who; //will be wired when
Components.wireVariables is called

    public Username() {
        //1. Render the template
        Executions.createComponents("/WEB-INF/composite/username.zul",
this, null);

        //2. Wire variables, components and event listeners (optional)
        Selectors.wireVariables(this, this, null);
        Selectors.wireComponents(this, this, false);
        Selectors.wireEventListeners(this, this);
    }

    public String getWho() {
        return mc_who.getValue();
    }

    public void setWho(String who) {
        mc_who.setValue(who);
    }

    //public void onOK() {...} //Add event listeners if required, and
wired by Components.addForwards
}
```

After org.zkoss.zk.ui.Component, java.util.Map) Executions.createComponents(java.lang.String, org.zkoss.zk.ui.Component, java.util.Map)^[3] is called, all components specified in the template will be instantiated

and become the child components of the composite component (Row). Notice that the URI must match the location of the template correctly.

Depending on the implementation you want, you could wire the data members (`mc_who`) by calling `java.lang.Object, boolean) Selectors.wireComponents(org.zkoss.zk.ui.Component, java.lang.Object, boolean)`^[5]. This method will search all data members and setter methods and *wire* the component with the same ID. Similarly, `java.lang.Object) Selectors.wireEventListeners(org.zkoss.zk.ui.Component, java.lang.Object)`^[6] is used to wire event listeners.

For more information, please refer to the Wire Components section and Wire Event the Listeners section sections.

Notice that there is a utility called ZK Composite^[7]. With the help of ZK Composite^[7], components are created and wired automatically based on the Java annotations you provide. In other words, Step 3 and 4 are done automatically. For more information, please refer to the Define Components with Java Annotations section.

Wire Spring-managed Beans

`java.lang.Object, java.util.List) Selectors.wireVariables(org.zkoss.zk.ui.Component, java.lang.Object, java.util.List)`^[4] will wire variables that can be resolved by the registered variable resolver. In addition to the variable-resolver directive, you can create any variable resolver manually and pass it as the third argument. `java.lang.Class) Selectors.newVariableResolvers(java.lang.Class, java.lang.Class)`^[8] provides a convenient way to instantiate variable resolvers. For example, let us say we'd like to wire Spring-manage beans, then we can do as follows.

```
@VariableResolver(org.zkoss.zkplus.spring.DelegatingVariableResolver)
public class Username extends Row implements IdSpace {
    @WireVariable
    private User user;

    public Username() {
        Executions.createComponents("/WEB-INF/composite/username.zul",
this, null);

        Selectors.wireVariables(this, this,
            Selectors.newVariableResolvers(getClass(), Row.class));
        Selectors.wireComponents(this, this, false);
        Selectors.wireEventListeners(this, this);
    }
    ...
}
```

`java.lang.Class) Selectors.newVariableResolvers(java.lang.Class, java.lang.Class)`^[8] will look for the `@VariableResolver` annotation and instantiate it automatically. As shown, we annotate `DelegatingVariableResolver`^[9] to resolve Spring-managed bean.

For more information, please refer to the Wire Variables section.

ID Space

Unless you extend a component that is an ID space owner (such as Window^[1]), all child components specified in the template will be in the same ID space as its parent. It might be convenient at the first glance. However, it will cause ID conflict if we have multiple instances of the same composite component. Thus, it is generally suggested to make the composite component a space owner.

It can be done easily by implementing an extra interface IdSpace^[1]. No other method needs to be implemented.

```
public class Username extends Row implements IdSpace {  
    ...  
}
```

Of course, if you prefer not to have an additional ID space, you don't need to implement IdSpace^[1].

Use Composite Component

Like macros and any other primitive components, you have to declare a composite component before using it. This can be done by using component directives. Then, we could use it the same way (they are actually primitive components). For example,

```
<?component name="username" extends="row" class="foo.Username"?>  
  
<grid>  
    <rows>  
        <username who="Joe"/>  
        <username who="Hellen"/>  
    </rows>  
</grid>
```

Define Composite Components as Standard Components

If a composite component is used in multiple pages, it is better to define it in the application level, such that it can be accessed in any page without any component directives.

There are basic two approaches to define a component in the application level:

1. Define it in an XML file which is called a language addon.
2. Define it with Java annotations.

Define Components in a Language Addon

A language addon is an XML file providing additional component definitions or customizing the standard components. For example, you can define the username component described in the previous section as follows.

```
<language-addon>  
    <addon-name>myapp</addon-name>  
    <component>  
        <component-name>username</component-name>  
        <extends>rows</extends>  
        <component-class>foo.Username</component-class>  
    </component>  
</language-addon>
```

For more information, please refer to Customization: Component Properties.

Define Components with Java Annotations

Instead of maintaining the definitions in the language addon as described above, you can define the component with Java annotation with a utility called ZK Composite^[10]. For example,

```
@Composite(name="username", macroURI="/WEB-INF/partial/username.zul")
public class Username extends Rows implements IdSpace {
    @Wire
    private Textbox mc_who; //will be wired when
Components.wireVariables is called

    //Note: no need to create components and wire variables/components

    public String getWho() {
        return mc_who.getValue();
    }
    public void setWho(String who) {
        mc_who.setValue(who);
    }
}
```

This approach is suggested if you have to develop several composite components. As shown, it is more convenient since you don't have to maintain a separate XML file (the language addon). Furthermore, it will create the components and wire them automatically based on the annotations.

Notice that it requires additional JAR files^[11], please refer to Small Talks: Define Composite Component using Java Annotation in ZK6 for the details.

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Row.html#>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Div.html#>
- [3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#createComponents\(java.lang.String,](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#createComponents(java.lang.String,)
- [4] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/select>Selectors.html#wireVariables\(org.zkoss.zk.ui.Component,](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/select>Selectors.html#wireVariables(org.zkoss.zk.ui.Component,)
- [5] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/select>Selectors.html#wireComponents\(org.zkoss.zk.ui.Component,](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/select>Selectors.html#wireComponents(org.zkoss.zk.ui.Component,)
- [6] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/select>Selectors.html#wireEventListeners\(org.zkoss.zk.ui.Component,](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/select>Selectors.html#wireEventListeners(org.zkoss.zk.ui.Component,)
- [7] <http://github.com/zanyking/ZK-Composite>
- [8] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/select>Selectors.html#newVariableResolvers\(java.lang.Class,](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/select>Selectors.html#newVariableResolvers(java.lang.Class,)
- [9] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkplus/spring/DelegatingVariableResolver.html#>
- [10] <https://github.com/zanyking/ZK-Composite>
- [11] <http://github.com/zanyking/ZK-Composite/downloads>

Client-side UI Composing

Though optional, you could have the total control of the client's functionality without the assistance of server-side coding. Generally, you don't need to do it. You don't even need to know how ZK Client Engine and client-side widgets communicate with the server. Their states can be synchronized automatically with ZK. However, you can still control this type of synchronization if you want. It is the so-called Server-client fusion.

A good rule of thumb is that you should handle events and manipulate UI mostly, if not all, on the server, since it is more productive. Then, you could improve the responsiveness and visual effects, and/or reduce 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.

- About client-side UI composing, please refer to ZK Client-side Reference: UI Composing.
- About customizing client-side widget's behavior, please refer to ZK Client-side Reference: Widget Customization.
- About client-side event handling, please refer to ZK Client-side Reference: Event Listening

Shadow for MVC

Introduction

In ZK 8.0.0, we have introduced shadow elements, such as a boilerplate code, to help application developers compose HTML layouts with dynamic data. It is inspired by Shadow DOM to enable better composition of ZK components. For more details, please check out our ZK MVVM Reference ^[1]. You can also use shadow elements with the MVC pattern; however, there are some differences. We will discuss this more in the following sections.

In the MVC pattern, developers can declare shadow tags in zul files, but the behavior is very different without MVVM annotation. For example,

```
<apply template="any" />
<template name="any">
  ...
</template>
```

The shadow element "apply" will not exist after the output is rendered to the client, so developers can't dynamically change the template content. For this purpose, we provide two kinds of Java classes for those who favor MVC:

- ShadowTemplate ^[2]
- CollectionTemplate ^[3]

They are NOT like the typical shadow elements defined in zul but components you can only create in Java code.

Setup

Before using shadow elements, make sure you include the required jar - zuti.jar. With maven, you should add the dependency below:

```
<dependency>
    <groupId>org.zkoss.zk</groupId>
    <artifactId>zuti</artifactId>
    <version>${zk.version}</version>
</dependency>
```

Wire Shadow Components

Like wiring a UI component, you can wire a shadow component.

Use ShadowTemplate

ShadowTemplate^[2] is a utility class that allows developers to apply shadow elements in Java class. It has a similar behavior to Apply^[4]; for example, developers can specify the template or pass parameters. The difference is that developers must designate a boolean value, called **autodrop**, to indicate whether to drop those rendered children or not. If true, every time the user changes the template or detaches from the original host, ShadowTemplate will HtmlShadowElement.recreate()^[5] or remove the children; otherwise, rendered children will remain. After instantiating ShadowTemplate instance, developers can trigger ShadowTemplate.apply(org.zkoss.zk.ui.Component)^[6] to compose the specified template, with shadow host passed as a parameter. Note: the passed host should be the same one if **autodrop** is true, or pass null to detach the original host first.

Note: ShadowTemplate doesn't support setting both template and template URI at the same time; one of them should be a null or empty string before setting another.

Example

Assume we have a zul file like this:

```
<zk>
    <div apply="DemoComposer">
        <div id="host1"></div>
    </div>
    <template name="labels">
        <label value="zul label"/>
        <x:label>xhtml label</x:label>
        <n:span>native span</n:span>
    </template>
</zk>
```

and in DemoComposer.java

```
@Wire
Div host1;

public void doAfterCompose(Component comp) throws Exception {
    super.doAfterCompose(comp);
```

```

    ShadowTemplate st = new ShadowTemplate(true); //autodrop = true
    st.setTemplate("labels");
    st.apply(host1);
}

```

- Line 6: we instantiate a new ShadowTemplate with **autodrop** which is equal to true.
- Line 7: assign the template name to st.
- Line 8: call apply method and shadow host is Div host1.

Then, we can see template "labels" are rendered and the created components are attached to host1.

If we have a button to change the template:

```

@Listen("onClick = #btn")
public void clickBtn() {
    st.setTemplate("othertemplate");
    st.apply(st.getShadowHost());
}

```

Those components rendered before will be detached first before attaching the new ones. Note: developers have to call `apply(host)` method again.

If developers want to apply other shadow hosts, please apply null first and then reapply like this:

```

st.apply(null);
st.apply(otherHost);

```

And the rendered components will also be detached.

Another case is when **autodrop** is equal to false. Here, neither changing the template nor applying other hosts (yes, you can apply whichever hosts you want) will cause rendered components to be detached.

Use CollectionTemplate

CollectionTemplate^[3] is similar to ShadowTemplate^[2]. The difference is that developers can assign ListModel^[7] and CollectionTemplateResolver^[8] for iterative rendering.

Example

The basic usage is simple. Here we demonstrate by using the previous sample code:

```

<zk>
    <div apply="DemoComposer">
        <div id="host1"></div>
    </div>
    <template name="labels">
        <label value="zul one ${each}"></label>
        <x:label>xhtml one ${each} </x:label>
        <n:span>native one ${each} </n:span>
    </template>
</zk>

```

The `each` in line 6, 7, 8 represents each item in ListModel, and in DemoComposer.java

```

@Wire
Div host1;
ListModel model = new ListModelList(Arrays.asList(new String[]{"1",
"2", "3"}));

public void doAfterCompose(Component comp) throws Exception {
    super.doAfterCompose(comp);
    CollectionTemplate ct = new CollectionTemplate(true); //autodrop
= true
    ct.setModel(model);
    ct.setTemplate("labels");
    ct.apply(host1);
}

```

Developers have to prepare a ListModel^[7] and assign it to the CollectionTemplate instance; they will then see that the template is created multiple times. Similarly, in cases where either the template or model is changed, apply(host) must be triggered for the effect to take place. The benefit of using CollectionTemplate is that every time the model's content changes, the layout will change as well, no matter if autodrop is true or false.

CollectionTemplateResolver

More advanced usage is to assign CollectionTemplateResolver^[8] to resolve template by evaluating the variable reference from model at runtime.

```

<zk>
    <div id="root" apply="DemoComposer">
        <div id="host1"></div>

        <template name="male">
            <div>
                <label>I'm male, my name is ${each.name}</label>
            </div>
        </template>
        <template name="female">
            <div>
                <label>I'm female, my name is ${each.name}</label>
            </div>
        </template>
    </div>
</zk>

```

The each in line 7, 12 represents each item in ListModel, and in DemoComposer.java

```

@Wire
Div host1;
ListModelList<Person> model = new ListModelList<Person>(new ArrayList<Person>() {
    add(new Person(true));
    add(new Person(false));
    add(new Person(false));
    add(new Person(true));
}

```

```

} });

public void doAfterCompose(Component comp) throws Exception {
    super.doAfterCompose(comp);
    CollectionTemplate ct = new CollectionTemplate(true); //autodrop
    = true
    ct.setModel(model);
    ct.setTemplateResolver(new MyCollectionTemplateResolver<Person>());
    ct.apply(host1);
}

public class MyCollectionTemplateResolver<E extends Person> implements
CollectionTemplateResolver<E> {
    public Template resolve(E o) {
        if (o.getGender())
            return root.getTemplate("male");
        else
            return root.getTemplate("female");
    }
}

public class Person {
    String name = "old name";
    boolean isMale = true;
    .... getter and setter
}

```

In this example, we assign a `CollectionTemplateResolver` instead of template name or URI, and you will see template "male" is rendered when the gender of `Person` variable is male. That means, `CollectionTemplate` provides not only `setTemplate` and `setTemplateURI` but also supports determining template dynamically by giving `CollectionTemplateResolver`^[8] like line 14, so the template will be resolved by evaluating the variable reference from model in runtime.

Because these 3 methods: `setTemplate()`, `setTemplateURI()` and `setTemplateResolver()`, serve the same purpose, please just call one of them. If you call them all, the later one will override the previous one.

Comparison

Although the behavior of `ShadowTemplate` and `Macro` component looks similar, there are some differences.

	ShadowTemplate	Macro Component
change host/parent	if autodrop is true, the rendered components will change parent; otherwise, they will stick with the same parent(or host).	doesn't matter if it is in-line or not; the rendered components will change parent.
change template/uri	if autodrop is true, the rendered components will be detached; otherwise, they will stick with the same parent(or host).	doesn't matter if it is in-line or not; the rendered components will be detached.

In short, while using Macro components, we would have to instantiate more than one to achieve this goal. ShadowTemplate has more flexibility for templating; with only one ShadowTemplate instance, developers can render anywhere without losing those rendered components. CollectionTemplate, too, can render template iteratively with ListModel, a task impossible for Macro component.

References

- [1] https://books.zkoss.org/zk-mvvm-book/10.0/shadow_elements/index.html
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zuti/zul/ShadowTemplate.html#>
- [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zuti/zul/CollectionTemplate.html#>
- [4] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zuti/zul/Apply.html#>
- [5] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlShadowElement.html#recreate\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlShadowElement.html#recreate())
- [6] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zuti/zul/ShadowTemplate.html#apply\(org.zkoss.zk.ui.Component\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zuti/zul/ShadowTemplate.html#apply(org.zkoss.zk.ui.Component))
- [7] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ListModel.html#>
- [8] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zuti/zul/CollectionTemplateResolver.html#>

Event Handling

An event (Event ^[1]) is used to abstract an activity made by a user, a notification made by an application, and an invocation of server push. Thus, the application can handle different kinds of notifications and sources with a universal mechanism. By and large, developers can even use the same approach to handle, say, message queues.

In this section we will discuss how to handle events, such as listening, posting and forwarding.

References

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

Event Listening

Listen by Use of an Event Listener

Event Listener

An event listener is a class implementing `EventListener`^[1]. For example,

```
public class MyListener implements EventListener {
    public void onEvent(Event event) {
        Messagebox.show("Hello");
    }
}
```

Then, you can register an event listener to the component that might receive the event by the use of `org.zkoss.zk.ui.event.EventListener` `Component.addEventListener(java.lang.String, org.zkoss.zk.ui.event.EventListener)`^[2]. For example,

```
button.addEventListener("onClick", new MyListener());
```

This is a typical approach to handle events. However, it is a bit tedious to register event listeners one-by-one if there are a lot of listeners. Rather, it is suggested to use a composer as described in the following section.

Composer and Event Listener Autowiring

With ZK Developer's Reference/MVC, you generally do not need to register event listeners manually. Rather, they can be registered automatically by the use of the auto-wiring feature of a composer. For example,

```
public class MyComposer extends SelectorComposer {
    @Listen("onClick = button#hi")
    public void showHi() {
        Messagebox.show("Hello");
    }
    @Listen("onClick = button#bye")
    public void showBye() {
        Messagebox.show("Bye");
    }
    @Listen("onOK = window#mywin")
    public void onOK() {
        Messagebox.show("OK pressed");
    }
}
```

As shown above, the method to listen is annotated with the `Listen`^[3] annotation using the event name followed by a selector string identifying the component(s) (for more selector syntax examples see `SelectorComposer`^[3]). The composer will register each annotated method as an event listener to the selected component automatically **in the same ID space**. Then, in the ZUL page, you can specify the `apply` attribute to associate the composer with a component.

```
<window id="mywin" apply="MyComposer">
    <textbox/>
    <button id="hi"/>
    <button id="bye"/>
</window>
```

If the listener needs to access the event, just declare it as the argument:

```
@Listen("onClick = button#hi")
public void showHi(MouseEvent event) {
    Messsagebox.show("Hello, " + event.getName());
}
```

Though not limited, a composer is usually associated with an ID space (such as Window^[1]) to handle events and components within the given ID space. You could associate any component that properly represents a scope of your application to manage.

For more information please refer to the Wire Event Listeners section.

Deferrable Event Listeners

By default, events are sent to the server when it is fired at the client. However, many event listeners are just used to maintain the status on the server, rather than providing visual responses to the user. In other words, there is no need to send the events for these listeners immediately. Rather, they shall be sent at once to minimize the traffic between the client and the server so as to improve the server's performance. For the sake of convenience, we call them the deferrable event listeners.

To make an event listener deferrable, you have to implement Deferrable^[4] (with EventListener) and return true for the isDeferrable method as follows.

```
public class DeferrableListener implements EventListener, Deferrable {
    private boolean _modified;
    public void onEvent(Event event) {
        _modified = true;
    }
    public boolean isDeferrable() {
        return true;
    }
}
```

When an event is fired at the client (e.g., the user selects a list item), ZK won't send the event if no event listener is registered for it or only deferrable listeners are registered. Instead, the event is queued at the client.

On the other hand, if at least one non-deferrable listener is registered, the event will be sent immediately with all queued events to the server at once. No event is lost and the arriving order is preserved.

Tip: Use the deferrable listeners for maintaining the server status, while the non-deferrable listeners for providing the visual responses for the user.

Page-level Event Listener

Developers could add event listeners to a page (Page^[8]) dynamically by org.zkoss.zk.ui.event.EventListener) Page.addEventListener(java.lang.String, org.zkoss.zk.ui.event.EventListener)^[5]. Once added, all events of the specified name sent to any components of the specified page will be sent to the listener.

All event listeners added to a page (so-called page-level event listeners) are assumed to be deferrable, no matter if Deferrable^[4] is implemented or not.

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

```
page.addEventListener("onChange", new EventListener() {
    public void onEvent(Event event) {
        modified = true;
    }
});
```

Listen by the use of an Event Handler

An event handler is a method specified as an event attribute of a ZK page or as a member of a component class.

Declare an Event Handler in ZUML

An event handler can be declared in a ZUL page by specifying an event attribute^[6]. For example,

```
<button label="hi" onClick='alert("Hello")'>
```

where the content of the event handler is the code snippet in Java. The event handler will be interpreted at run time (by use of BeanShell). If you prefer to use another language, you could specify the language name in front of it. For example, the following uses Groovy as the interpreter:

```
<button label="hi" onClick="groovy:alert('Hi, Groovy')">
```

Important Builtin Variables

- self - the component receiving the event. In the previous example, it is the button itself.
- event - the event being received. In the previous example, it is an instance of MouseEvent^[7].

Notice that the event handler declared in this way is interpreted at run time, so it inherits all advantages and disadvantages of interpreter-based execution.

Advantages:

- It can be changed on the fly without recompiling and reloading the application.
- Easy to maintain if the code snippet is small.

Disadvantages:

- Slower to run.
- Compilation errors can not be known in advance.
- Hard to maintain if mixing business logic with user interface.

Suggestion:

- It is generally suggested to use this approach for 1) prototyping, or 2) simple event handling.

Declare an Event Handler in Java

The other way to have an event handler is to declare it as a member of a component class. For example,

```
public class MyButton extends Button {  
    public void onClick() {  
        Messagebox.show("Hello");  
    }  
}
```

If the event handler needs to handle the event, it can declare the event as the argument as follows.

```
public class MyButton extends Button {  
    public void onClick(MouseEvent event) {  
        Messagebox.show("Hello, "+event.getName());  
    }  
}
```

Suggestions:

- It is suggested to use this approach for component development, since it is subtle for application developers to notice its existence. In addition, it requires to extend the component class.

-
- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/EventListener.html#>
 - [2] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#addEventListener\(java.lang.String,java.util.EventListener\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#addEventListener(java.lang.String,java.util.EventListener))
 - [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/select/annotation/Listen.html#>
 - [4] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Deferrable.html#>
 - [5] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Page.html#addEventListener\(java.lang.String,java.util.EventListener\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Page.html#addEventListener(java.lang.String,java.util.EventListener))
 - [6] An event attribute is an attribute starting with `on`
 - [7] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/MouseEvent.html#>

Precedence of Listeners

The order of precedence for listeners from the highest to the lowest is as follows.

1. Event listeners implemented with Express (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Express.html#>), and registered by `org.zkoss.zk.ui.event.EventListener` `Component.addEventListener(java.lang.String, org.zkoss.zk.ui.event.EventListener)` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#addEventListene>(`java.lang.String`,`org.zkoss.zk.ui.event.EventListener`))
2. Event handlers defined in a ZUML document
3. Event listeners registered by `org.zkoss.zk.ui.event.EventListener` `Component.addEventListener(java.lang.String, org.zkoss.zk.ui.event.EventListener)` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#addEventListene>(`java.lang.String`,`org.zkoss.zk.ui.event.EventListener`)) (and without Express (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Express.html#>))
 - It includes the method of a composer wired by `GenericForwardComposer` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/GenericForwardComposer.html#>) because the event listener is used.
4. Event handlers defined as a class's method
5. Event listeners registered to a page by `org.zkoss.zk.ui.event.EventListener` `Page.addEventListener(java.lang.String, org.zkoss.zk.ui.event.EventListener)` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Page.html#addEventListene>(`java.lang.String`,`org.zkoss.zk.ui.event.EventListener`))

Abort the Invocation Sequence

You could abort the calling sequence by calling `Event.stopPropagation()` ([`http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Event.html#stopPropagation\(\)`](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Event.html#stopPropagation())). Once one of the event listeners invokes this method, all the following event handlers and listeners are ignored.

Version History

Version	Date	Content
5.0.6	November 2010	SerializableEventListener (<code>http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/SerializableEventListener.html#</code>) was introduced to simplify the instantiation of a serializable anonymous class

Event Firing

Events are usually fired (aka., triggered) by a component (when serving the user at the client). However, applications are allowed to fire events too.

There are three ways to trigger an event: post, send and echo.

Post an Event

Posting is the most common way to trigger an event. By posting, the event is placed at the end of the system event queue^[1]. Events stored in the system event queue are processed one-by-one in first-in-first-out order. Each desktop has one system event queue and all events are handled sequentially.

To trigger an event, you could invoke `org.zkoss.zk.ui.Component, java.lang.Object` `Events.postEvent(java.lang.String, org.zkoss.zk.ui.Component, java.lang.Object)`^[2]. For example,

```
Events.postEvent("onClick", button, null); //simulate a click
```

In addition to posting an event to the end of the system event queue, you could specify a priority with `java.lang.String, org.zkoss.zk.ui.Component, java.lang.Object` `Events.postEvent(int, java.lang.String, org.zkoss.zk.ui.Component, java.lang.Object)`^[3]. By default, the priority is 0. The higher the priority the earlier an event is processed.

Notice that the invocation returns after placing the event in the system event queue. In other words, the event won't be processed unless all other events posted earlier or with higher priority are processed.

[1] Please don't confuse it with the event queues discussed in the event queues section, which are application-specific, while the system event queue is invisible to application developers.

[2] [`http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Events.html#postEvent\(java.lang.String, org.zkoss.zk.ui.Component, java.lang.Object\)`](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Events.html#postEvent(java.lang.String, org.zkoss.zk.ui.Component, java.lang.Object)),

[3] [`http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Events.html#postEvent\(int, java.lang.String, org.zkoss.zk.ui.Component, java.lang.Object\)`](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Events.html#postEvent(int, java.lang.String, org.zkoss.zk.ui.Component, java.lang.Object)),

Send an Event

If you prefer to trigger an event to a component directly and process it immediately, rather than placing it in the system event queue and waiting for execution, you could use `org.zkoss.zk.ui.Component, java.lang.Object` `Events.sendEvent(java.lang.String, org.zkoss.zk.ui.Component, java.lang.Object)` ([`http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Events.html#sendEvent\(java.lang.String, org.zkoss.zk.ui.Component, java.lang.Object\)`](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Events.html#sendEvent(java.lang.String, org.zkoss.zk.ui.Component, java.lang.Object))) to trigger the event.

```
Events.sendEvent("onMyEvent", component, mydata);
```

org.zkoss.zk.ui.Component, java.lang.Object) Events.sendEvent(java.lang.String, org.zkoss.zk.ui.Component, java.lang.Object) ([`http:// www. zkoss. org/ javadoc/ latest/ zk/ org/ zkoss/ zk/ ui/ event/ Events.html#sendEvent\(java.lang.String\)`](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Events.html#sendEvent(java.lang.String))) won't return until all handlers and listeners registered for this event have been processed. You could image it as a method of invocation. Also notice that the event handlers and listeners are invoked directly without starting any event threads (no matter whether the event thread is enabled or not^[1]).

[1] By default, the event thread is disabled. Please refer to the Event Threads section for more information.

Echo an Event

Echoing is a way to delay event processing until the next AU request (aka., Ajax) is received.

More precisely, the event being echoed won't be queued into the system event queue. Rather, it asks the client to send back an AU request immediately. Furthermore, after the server receives the AU request, the event is then posted to the system event queue for processing.

In other words, the event won't be processed in the current execution. Rather, it is processed in the following request when the event is *echoed* back from the client. Here is an example of using org.zkoss.zk.ui.Component, java.lang.Object) Events.echoEvent(java.lang.String, org.zkoss.zk.ui.Component, java.lang.Object) ([`http:// www. zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Events.html#echoEvent\(java.lang.String\)`](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Events.html#echoEvent(java.lang.String))):

```
Events.echoEvent("onMyEvent", component, mydata);
```

Event echoing is useful for implementing a long operation. HTTP is a request-and-response protocol, so the user won't receive any feedback until the request has been served and responded. Thus, we could send back some busy messages to let the user know what has happened, and echo back an event to do the long operation. For more information, please refer to the Long Operations: Use Echo Events section.

Event Forwarding

Overview

For easy programming, ZK does not introduce any complex event flow. When an event is sent to a target component, only the event listeners registered for the target component will be called. It is the application's job to forward an event to another component if necessary.

For example, you might have a menu item and a button to trigger the same action, say, opening a dialog, and then it is more convenient to have a single listener to open the dialog, and register the listener to the main window rather than register to both the menu item and button.

Event Forwarding in Java

Forwarding an event is straightforward: just posting or sending the event again. However, there is a better way: composer. The composer can be the central place to handle the events. For example, you could invoke `openDialog` in the event handler for the menu item and button as shown below:

```
public class FooComposer extends SelectorComposer {  
    @Listen("onClick = menuitem#item1; onClick = button#btn")  
    private void openDialog() {  
        //whatever you want  
    }  
}
```

Event Forwarding in ZUML

Event forwarding can be done with the `forward` attribute in ZUML. For example,

```
<window id="mywin">  
    <button label="Save" forward="onSave"/>  
    <button label="Cancel" forward="onCancel"/>  
</window>
```

Then, `window` will receive the `onSave` event when the Save button is clicked.

With this approach we could introduce an abstract layer between the event and the component. For example, `window` needs only to handle the `onSave` event without knowing which component causes it. Therefore, you could introduce another UI to trigger `onSave` without modifying the event listener. For example,

```
<menuitem label="Save" forward="onSave"/>
```

Of course, you can use the composer and ZUML's `forward` together to have more maintainable code.

```
public class BetterComposer  
extends org.zkoss.zk.ui.select.SelectorComposer {  
    @Listen("onSave = #mywin")  
    public void doSave(ForwardEvent event) { //signature if you care  
        about the event  
        ...  
    }  
}
```

```

@Listen("onCancel = #mywin")
public void doCancel() { //signature if you don't care about the
event
...

```

Notice that, as shown above, the event being forwarded is wrapped as an instance of `ForwardEvent`^[1]. To retrieve the original event, you could invoke `ForwardEvent.getOrigin()`^[2].

Using a component Path

You can also use a component Path^[3] within your ZUML pages to specify a target component to which you would like to forward a specific event. This is especially useful if you want to forward events across different IdSpace^[4] such as forwarding events from a component in an included ZUML page to the main page component. For example,

```

<?page id="mainPage" ?>
<window id="mainWindow" apply="BetterComposer">
...
<include src="incDetails.zul" />
...
</window>

```

Now in your included page use Path^[3] while forwarding events to `mainWindow Window` component.

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

Forward with Parameters

You can specify any application-specific data in the `forward` attribute by surrounding it with the parenthesis as shown below:

```

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

```

Then, you can retrieve the application-specific data by `ForwardEvent.getData()`^[5].

Notice: When using `forward` attribute in the ZUML(.zul) with ZK MVC controller, you have to get the original event by using `getOrigin()`, then you can access the data by `getData()`

- Example : ZUL

```

<tabbox id="ctrl" apply="composer1">
<tabs>
    <tab id="tb1" label="News" forward="ctrl.onSelectTab(0)"></tab>
    <tab id="tb2" label="News Images" forward="ctrl.onSelectTab(1)"></tab>
</tabs>
</tabbox>

```

- Example Composer (composer1)

```

@Listen("onSelectTab = #ctrl")
public void doChangeTab(ForwardEvent e) {
    MouseEvent me = (MouseEvent) e.getOrigin();
    System.out.println(me.getData());
}

```

Foward Multiple Events

If you want to forward several events at once, you can specify them in the forward attribute by separating them with commas , . For example,

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

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

References

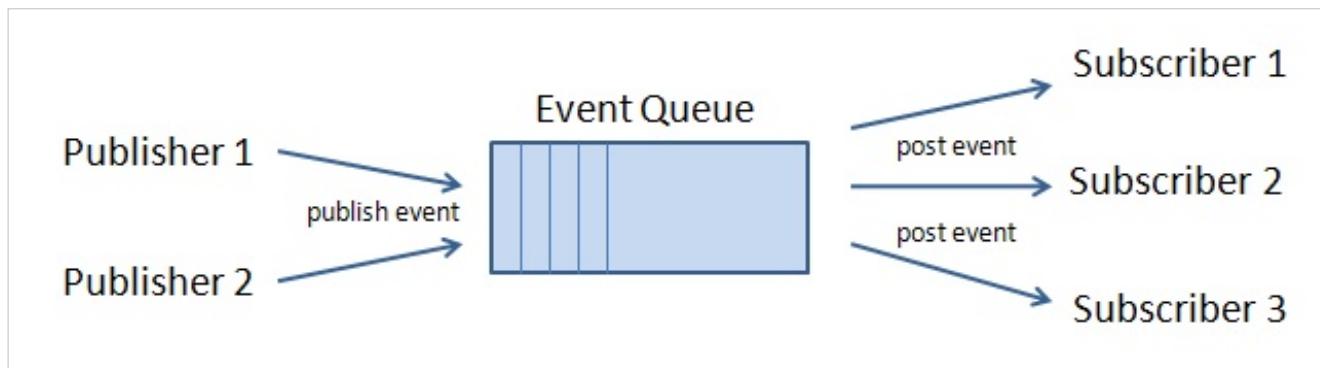
- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/ForwardEvent.html#>
- [2] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/ForwardEvent.html#getData\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/ForwardEvent.html#getData())
- [3] http://books.zkoss.org/wiki/ZK_Developer's_Guide/ZK_in_Depth/Component_Path_and_A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

Event Queues

Overview

An event queue is an event-based publish-subscriber solution for application information delivery and messaging. It provides asynchronous communications for different modules/roles in a loosely-coupled and autonomous fashion.

By publishing, a module (publisher) sends out messages without explicitly specifying or having knowledge of intended recipients. By subscribing, a receiving module (subscriber) receives messages that the subscriber has registered an interest in it, without explicitly specifying or knowing the publisher.



The purpose of event queues are two folds:

1. Simplify the many-to-many communication.
2. Make the application independent of the underlying communication mechanism. The application remains the same, while the event queue can be implemented with the use of Ajax, server push and even message queue.

Identification of an Event Queue

An event queue is identified by a name and a scope. The scope represents the visibility of an event queue. For example, while a desktop-scoped event queue is visible in the same desktop, the application-scoped event queue is only visible in the whole application.

Locate an Event Queue

You could locate an event queue by invoking one of the `lookup` methods of `EventQueues`^[1]. For example,

```
EventQueues.lookup("myQueue"); //assumes the desktop scope
EventQueues.lookup("anotherQueue", EventQueues.SESSION, true);
EventQueues.lookup("anotherQueue", session, true);
```

Notice that if you want to locate an event queue in a working thread (rather than an event listener), you have to use `org.zkoss.zk.ui.Session` `EventQueues.lookup(java.lang.String, org.zkoss.zk.ui.Session)`^[2] or `org.zkoss.zk.ui.Application`) `EventQueues.lookup(java.lang.String, org.zkoss.zk.ui.Application)`^[2], depending on your requirement.

The Scope of an Event Queue

There are currently four different scopes: desktop, group, session, and application. In addition, you add your own scope, such as that you can include a message queue to communicate among several servers.

Scope	API	Description
desktop	<code>java.lang.String)</code> <code>EventQueues.lookup(java.lang.String, java.lang.String)</code> ^[2] <code>java.lang.String, boolean)</code> <code>EventQueues.lookup(java.lang.String, java.lang.String, boolean)</code> ^[2]	The event queue is visible only in the same desktop. (DesktopEventQueue ^[3])
group	<code>java.lang.String)</code> <code>EventQueues.lookup(java.lang.String, java.lang.String)</code> ^[2] <code>java.lang.String, boolean)</code> <code>EventQueues.lookup(java.lang.String, java.lang.String, boolean)</code> ^[2]	<ul style="list-style-type: none"> Available for ZK: CE PE EE <p>The event queue is visible only in a group of desktops that belongs to the same browser tab(page). It is formed if an iframe or frameset is used. Some portal containers might cause a group of desktops to be formed too. Unlike the session and application scope, the group scope doesn't require the server push, so the communication is more efficient.</p>
session	<code>java.lang.String)</code> <code>EventQueues.lookup(java.lang.String, java.lang.String)</code> ^[2] <code>java.lang.String, boolean)</code> <code>EventQueues.lookup(java.lang.String, java.lang.String, boolean)</code> ^[2] <code>org.zkoss.zk.ui.Session)</code> <code>EventQueues.lookup(java.lang.String, org.zkoss.zk.ui.Session)</code> ^[2] <code>org.zkoss.zk.ui.Session, boolean)</code> <code>EventQueues.lookup(java.lang.String, org.zkoss.zk.ui.Session, boolean)</code> ^[2]	<p>The event queue is visible only in the same session. The server push will be enabled automatically if it subscribes a session-scoped event queue.</p> <p>Notice that the server push is disabled automatically if the current desktop doesn't subscribe to any session- or application-scoped event queue. Also, notice that the locating and creating of an event queue and publishing an event won't start the server push.</p> <p>ZK 5.0.5 and Prior: When a server push is enabled, a working thread is instantiated and started. It means this feature cannot be used in an environment that doesn't allow working threads, such as Google App Engine. No such limitation is likely to occur in ZK 5.0.6 or later.</p>

application	<pre>java.lang.String) EventQueues.lookup(java.lang.String, java.lang.String) [2] java.lang.String, boolean) EventQueues.lookup(java.lang.String, java.lang.String, boolean) [2] org.zkoss.zk.ui.WebApp) EventQueues.lookup(java.lang.String, org.zkoss.zk.ui.WebApp) [2] org.zkoss.zk.ui.WebApp, boolean) EventQueues.lookup(java.lang.String, org.zkoss.zk.ui.WebApp, boolean) [2]</pre>	<p>The event queue is visible only in the whole application. The server push will be enabled automatically.</p> <p>Notice that the server push is disabled automatically if the current desktop doesn't subscribe to any session- or application-scoped event queue. Also notice that the locating and creating of an event queue and publishing an event won't start the server push.</p> <p>ZK 5.0.5 and Prior: When a server push is enabled, a working thread is instantiated and started. It means this feature cannot be used in an environment that doesn't allow working threads, such as Google App Engine. No such limitation is likely to occur in ZK 5.0.6 or later.</p>
-------------	--	---

Here is a summary of the differences.

	desktop	group	session	application
visibility	desktop	group	session	application
publish	only in an event listener, or the current execution is available.	only in an event listener, or the current execution is available.	no limitation	no limitation
subscribe	only in an event listener, or the current execution is available.	only in an event listener, or the current execution is available.	only in an event listener, or the current execution is available.	only in an event listener, or the current execution is available.
multi-thread	Not used	Not used	5.0.5 or prior: Used (transparent) 5.0.6 or later: Not used	5.0.5 or prior: Used (transparent) 5.0.6 or later: Not used
server-push	Not used*	Not used*	Used (transparent)	Used (transparent)
Cluster Environment	Support	Support	Support	Unsupported (Java Spec. Limitation)
Availability	<ul style="list-style-type: none"> Available for ZK: 	<ul style="list-style-type: none"> Available for ZK: 	<ul style="list-style-type: none"> Available for ZK: 	<ul style="list-style-type: none"> Available for ZK:
Use Cases	<ul style="list-style-type: none"> send events between different pages within one desktop 		<ul style="list-style-type: none"> a user logs in one browser tab, log out the same user in other browser tabs of the same session 	<ul style="list-style-type: none"> send messages in a chatroom application broadcast system messages to all users/desktops

- If you register an asynchronous listener, it still enables server-push.

Publish and Subscribe

Publish an Event

To publish, just invoke one of the `publish` methods of `EventQueue`^[4] (returned by `lookup`). For example,

```
EventQueues.lookup("my super queue", EventQueues.APPLICATION, true)
    .publish(new Event("onSomethingHappening", null, new
SomeAdditionInfo()));
```

The message used to communicate among publishers and subscribers is `Event`^[1] object, so you can use any of its subclasses you prefer.

Subscribe with a Synchronous Event Listener

An event queue will broadcast a published event to all subscribers by calling each subscriber's event listener. To subscribe an event queue with a synchronous listener, just invoke `EventQueue.subscribe`^[5] (returned by `lookup`). The event listener is invoked just like a normal event. You can manipulate ZK components, update UI by notifying a bean's change or implement your business logic. For example:

```
EventQueues.lookup("my super queue", EventQueues.APPLICATION,
true).subscribe(
    new EventListener() {
        public void onEvent(Event evt) {
            //handle the event just like any other event listener, can
            change components' state
        }
    });
});
```

Asynchronous Event Listener

By default, the subscribed event listeners are invoked the same way as invocations of the listeners for a posted event. They are invoked one-by-one. No two event listeners belonging to the same desktop will be invoked at the same time. In addition, it is invoked under an execution (i.e., `Executions.getCurrent`^[6] is never null). It is allowed to manipulate the components belonging to the current execution. For the sake of description, we call them synchronous event listeners.

On the other hand, the event queue also supports the so-called asynchronous event listener, which is invoked **asynchronously in another thread**. There is **no current execution** available. It is **not** allowed to access any component. It is designed to execute a long operation without blocking users from accessing other UI in a browser.

For more information and examples, please refer to the Long Operations: Use Event Queue section.

Clean Up

Remember to unsubscribe [7] to an event queue with a listener when you don't need it anymore to avoid performing duplicate actions.

It's better to remove an event queue manually as you don't need it. If you create a **desktop** scope event queue, since it's stored as a desktop's attribute, it will be destroyed when the desktop is destroyed. The same rule applies to a session-scoped event queue.

Extend Event Queue: Add a Custom Scope

The location and creation of an event queue are actually done by a so-called event queue provider. An event-queue provider must implement the `EventQueueProvider` [8] interface.

To customize it, just provide an implementation, and then specify the class name in the property called `org.zkoss.zk.ui.event.EventQueueProvider.class`.

For example, let us say we want to introduce the JMS scope, then we can implement as follows (only pseudo-code):

```
public class MyEventQueueProvider extends
org.zkoss.zk.ui.event.impl.EventQueueProviderImpl {
    public EventQueue lookup(String name, String scope, boolean
autocreate) {
        if ("jms".equals(scope)) {
            //create an event queue based on JMS's name
        } else
            return super.lookup(name, scope, autocreate);
    }
    public boolean remove(String name, String scope) {
        if ("jms".equals(scope)) {
            //remove the event queue based on JMS's name
        } else
            return super.remove(name, scope);
    }
}
```

Then, specify the property in `WEB-INF/zk.xml`

```
<library-property>
    <name>org.zkoss.zk.ui.event.EventQueueProvider.class</name>
    <value>MyEventQueueProvider</value>
</library-property>
```

Event Queues and Server Push

When an application-scope event queue is used, the server push is enabled automatically for each desktop that subscribers belong to. In addition, a thread is created to forward the event to subscribers.

ZK supports several server push implementations and configurations including client-polling and comet^[9], please refer to Server_Push/Configuration.

Use Case

Application-scope: Chatroom

Here is an example: chat.

```
<window title="Chat" border="normal">
    <zscript><![CDATA[
        import org.zkoss.zk.ui.event.*;
        EventQueue que = EventQueues.lookup("chat",
EventQueues.APPLICATION, true);
        que.subscribe(new EventListener() {
            public void onEvent(Event evt) {
                new Label(evt.getData()).setParent(inf);
            }
        });
        void post(Textbox tb) {
            String text = tb.value;
            if (text.length() > 0) {
                tb.value = "";
                que.publish(new Event("onChat", null, text));
            }
        }
    ]]></zscript>

    Say <textbox onOK="post(self)" onChange="post(self)" />
    <separator bar="true"/>
    <vbox id="inf"/>
</window>
```

Then, you can chat among two or more different computers.

Desktop scope: among Multiple ZUL Pages

It is a typical approach to split page layout with several ZUL pages, and EventQueue could also be used for communicating between these pages. Here is an example for using EventQueue to interact between two composers. Let's say you have a main.zul page that is actually divided into two: page1.zul and page2.zul.

main.zul

```
<zk>
    <include src="page1.zul"></include>
    <include src="page2.zul"></include>
</zk>
```

page1.zul

```
<window title="ZUL page 1" border="normal" apply="demo.WindowComposer1">
    <button id="btn" label="change label in ZUL page 2" />
</window>
```

```
package demo;
public class WindowComposer1 extends SelectorComposer {

    private EventQueue eq;
    @Wire
    Button btn;

    @Listen("onClick = button#btn")
    public void changeLabel() {
        eq = EventQueues.lookup("interactive", EventQueues.DESKTOP,
true);
        eq.publish(new Event("onButtonClick", btn, "label is
Changed!"));
    }
}
```

page2.zul

```
<window title="ZUL page 2" border="normal" apply="demo.WindowComposer2">
    <label id="lbl" value="label in ZUL page 2" />
</window>
```

```
package demo;
public class WindowComposer2 extends SelectorComposer {

    private EventQueue eq;
    @Wire
    private Label lbl;

    public void doAfterCompose(Component comp) throws Exception {
        super.doAfterCompose(comp);
        eq = EventQueues.lookup("interactive", EventQueues.DESKTOP,
true);
    }
}
```

```
    eq.subscribe(new EventListener() {
        public void onEvent(Event event) throws Exception {
            String value = (String)event.getData();
            lbl.setValue(value);
        }
    });
}
```

By doing this, you can change data in ZUL page 2 by clicking the button in ZUL page 1. Note: Open main.zul page in browser to run the above sample codes. If there are any other ZUL pages that have subscribed to the same name in the EventQueues, the content in those ZUL pages will also be updated.

Version History

Version	Date	Content
5.0.4	August 2010	The group scope was introduced to allow the communication among inline frames without Server Push (minimizing the network bandwidth consumption).
5.0.6	November 2010	The event queue won't start any working threads and they are serializable, so it is safe to use them in a clustering environment.

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/EventQueues.html#>
 - [2] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/EventQueues.html#lookup\(java.lang.String,java.util.List\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/EventQueues.html#lookup(java.lang.String,java.util.List))
 - [3] <https://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/impl/DesktopEventQueue.html>
 - [4] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/EventQueue.html#>
 - [5] [https://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/EventQueue.html#subscribe\(org.zkoss.zk.ui.event.EventListener,java.util.List\)](https://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/EventQueue.html#subscribe(org.zkoss.zk.ui.event.EventListener,java.util.List))
 - [6] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#getCurrent>
 - [7] [https://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/EventQueue.html#unsubscribe\(org.zkoss.zk.ui.event.EventListener\)](https://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/EventQueue.html#unsubscribe(org.zkoss.zk.ui.event.EventListener))
 - [8] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/impl/EventQueueProvider.html#>
 - [9] [http://en.wikipedia.org/wiki/Comet_\(programming\)](http://en.wikipedia.org/wiki/Comet_(programming))

Client-side Event Listening

Overview

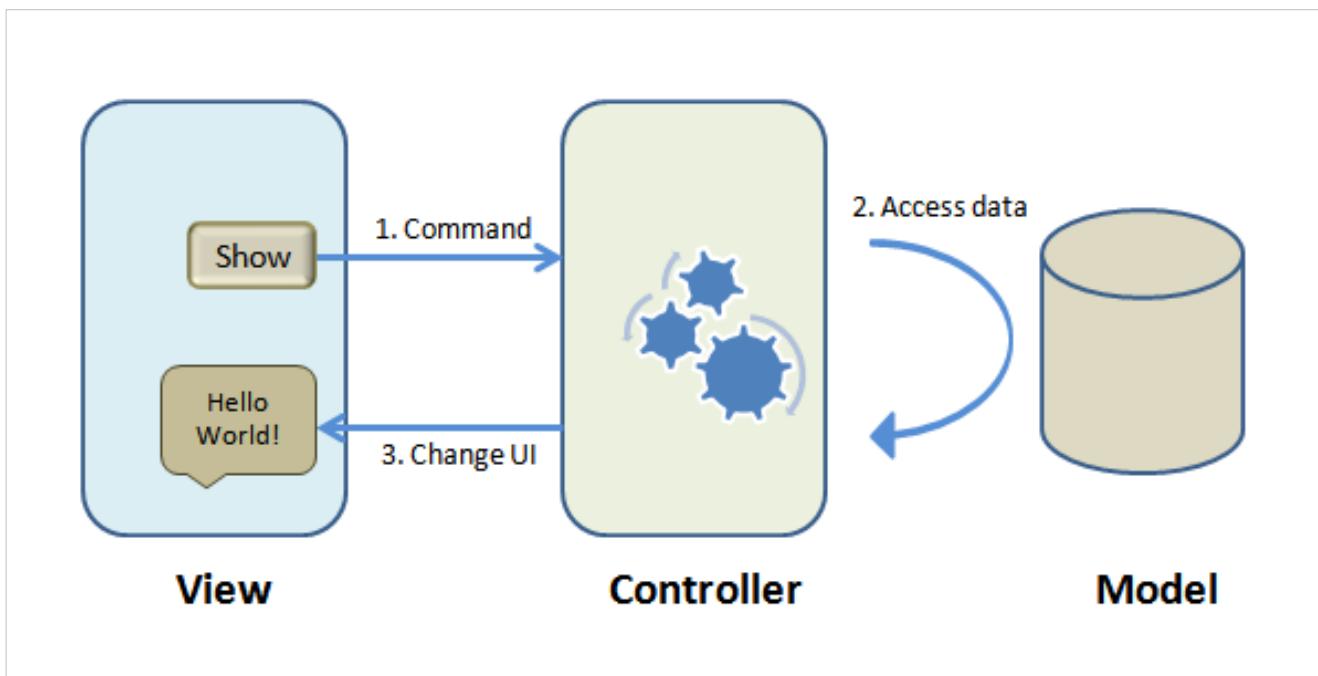
ZK allows applications to handle events at both the server and client side. Handling events at the server side, as described in the previous sections, is more common, since the listeners can access the backend services directly. However, handling events at the client side improves the responsiveness.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

A good 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.

For more information about handling events at the client, please refer to [ZK Client-side Reference: Event Listening](#).

MVC

MVC (Model-View-Control), more precisely, it is known as MVP (Model-View-Presenter) is a design pattern designed to separate the model, view, and controller. It is strongly suggested to apply the MVC pattern to your application, not only because it is easy to develop and maintain, but also the performance is great.



Alternative: MVVM

MVVM represents **Model**, **View**, and **ViewModel**. MVVM is identical to the Presentation Model^[1] introduced by Martin Fowler. It is a variant of the MVC design pattern. Unlike MVC, the control logic is implemented in a POJO class called the *view model*. It provides the further abstraction that a view model assumes *nothing* about any visual element in the view. It thus avoids mutual programming ripple effects between UI and the view model. On the other hand, some developers might find it not as intuitive as MVC. For more information, please refer to [MVVM Reference](#)^[2].

View

The *view* is UI -- a composition of components. As described in the UI Composing section, UI can be implemented by a ZUML document or in Java. For the sake of description, ZUML is used to illustrate the concept and features.

Controller

The *controller* is a Java program that is used to glue UI (view) and Data (model) together.

For a simple UI, there is no need to prepare a controller. For example, the data of a Listbox^[3] could be abstracted by implementing ListModel^[7].

For typical database access, the glue logic (i.e., control) can be handled by a generic feature called Data Binding. In other words, the read and write operations can be handled automatically by a generic Data Binding, and you don't need to write the glue logic at all.

To implement a custom controller, you could extend from SelectorComposer^[3], or implement Composer^[2] from scratch. Then, specify it in the element it wants to handle in a ZUML document.

Model

The *model* is the data an application handles. Depending on the application requirement, it could be anything as long as your controller knows it. Typical objects are POJOs, beans, Spring-managed beans, and DAOs.

In addition to handling the data in a controller, some components support the abstraction model to uncouple UI and data. For example, grid, listbox and combobox support ListModel^[7], while tree supports TreeModel^[4].

References

- [1] <http://martinfowler.com/eaaDev/PresentationModel.html>
 - [2] <http://books.zkoss.org/zk-mvvm-book/9.5/index.html>
 - [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Listbox.html#>
 - [4] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/TreeModel.html#>
-

Controller

Overview

The *controller* is a Java program that is used to glue UI (view) and Data (model) together.

A simple UI does not require any controllers. For example, the data of a Listbox^[3] could be abstracted by implementing ListModel^[7] as described in the Model section.

For typical database access, the glue logic (i.e., controller) can be handled by a generic feature called Data Binding. In other words, the create, read, update and delete operations (CRUD) can be handled automatically by a generic Data Binding mechanism, and you don't need to write the glue logic at all as described in the Data Binding section.

If none of the above fulfills your requirement, you could implement a custom controller (which is called a composer in ZK terminology). In the following sections we will discuss how to implement a custom controller in details.

Composer

Custom Controller

A custom controller is called a composer in ZK. To implement it, you can simply extend SelectorComposer^[3]. Then, specify it in the UI element that it wants to handle in a ZUML document.

A composer usually does, but not limited to:

- Load data to components, if necessary.
- Handle events and manipulate components accordingly, if necessary.
- Provide the data, if necessary.

In addition, a composer can be used to involve the lifecycle of ZK Loader for doing:

- Exception handling
- Component instantiation monitoring and filtering

A composer can be configured as a system-level composer, such that it will be called each time a ZUML document is loaded.

Implement Composers

To simplify the implementation of the controller part of UI, ZK provides several skeleton implementations. For example, SelectorComposer^[3], as one of the most popular skeletons, wires components, variables and event listeners automatically based on Java annotations you specify. For example, in the following controller and zul,

Controller:

```
package foo;
import org.zkoss.zk.ui.select.SelectorComposer;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.Listen;
import org.zkoss.zul.*;

public class MyComposer extends SelectorComposer<Window> {
```

```
@Wire  
Textbox input;  
  
@Wire  
Label output;  
  
@Listen("onClick=#ok")  
public void submit() {  
    output.setValue(input.getValue());  
}  
@Listen("onClick=#cancel")  
public void cancel() {  
    output.setValue("");  
}  
}
```

- Line: 9-12: The member fields `input`, `output` are automatically assigned with components with identifiers of "input" and "output", respectively.
- Line 14-21: The methods `submit()` and `cancel()` will be called when user clicks on the corresponding buttons.

ZUL:

```
<window apply="foo.MyComposer">  
    <div>  
        Input: <textbox id="input" />  
    </div>  
    <div>  
        Output: <label id="output" />  
    </div>  
    <button id="ok" label="Submit" />  
    <button id="cancel" label="Clear" />  
</window>
```

In addition to wiring components via identifiers, you could wire by a CSS3-like selector (Selector^[1]), such as

- `@Wire("#foo")`
- `@Wire("textbox, intbox, decimalbox, datebox")`
- `@Wire("window > div > button")`
- `@Listen("onClick = button[label='Clear'])")`

For more information, please refer to the following sections: Wire Components, Wire Variables and Wire Event Listeners.

Apply Composers

Once a composer is implemented, you usually associate it with a component, so that the composer can control the associated components and its child components.

Associating a composer to a component is straightforward: just specify the class to the apply attribute of the XML element you want to control. For example,

```
<grid apply="foo.MyComposer">
  <rows>
    <row>
      <textbox id="input"/>
      <button label="Submit" id="submit"/>
      <button label="Reset" id="reset"/>
    </row>
  </rows>
</grid>
```

Applying Multiple Composers

You could specify multiple composers; just separate them with commas. They will be called from left to right.

```
<div apply="foo.Composer1, foo2.Composer2">
```

Apply Composer Instances

In addition to the class name, you could specify an instance too. For example, suppose you have an instance called `fooComposer`, then

```
<grid apply="${fooComposer}">
```

If a class name is specified, each time the component is instantiated, an instance of the specified composer class is instantiated too. Thus, you don't have to worry about the concurrency issue. However, if you specify an instance, it will be used directly. Thus, you have to either create an instance for each request, or make it thread-safe.

Retrieve Composer in EL Expressions

If you have to retrieve the composer back later (such as reference it in an EL expression), you can store the composer into a component's attribute^[2].

If the composer extends from one of ZK skeletal implementations (such as `SelectorComposer`^[3] and `GenericForwardComposer`^[3]), it will be stored into an attribute automatically. Thus, for the sake of convenience, you could extend from one of these classes, if you'd like to retrieve the composer back.

Every ZK skeletal implementation provides several ways to name the composer as described in the following sections.

-
- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/select/Selector.html#>
 - [2] It can be done by invoking SelectorComposer (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/select/SelectorComposer.html#>), because the component's attribute can be referenced directly in EL expressions. Notice that if you want to reference it in EL expressions, you'd set the attribute in because UNIQ-javadoc-2-736dd3f9533a2ca2-QINU was called after all child components were instantiated.
 - [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/GenericForwardComposer.html#>

Default Names of Composer

If a composer extends from one of ZK skeletal implementations (such as SelectorComposer (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/select/SelectorComposer.html#>) and GenericForwardComposer (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/GenericForwardComposer.html#>)), the composer is stored in three component attributes called:

- `$composer`
- `id$composer`
- `id$ClassName`, where `id` is the component's ID, and `ClassName` is the class name of the composer. If ID is not assigned, it defaults to an empty string, so the composer will be stored to two component attributes:
`$composer` and `$ClassName`.

Therefore, you can access the composer with one of the above variables e.g.

```
<window id="mywin" apply="MyComposer">
    <textbox value="${mywin$composer.title}" />
    <textbox value="${$composer.title}" /> <!-- also refer to MyComposer -->
</window>
```

Notice that `$composer` is always assigned no matter what the ID is, so it is more convenient to use. However, if there are several components assigned with composers, you might have to use ID to distinguish them.

The second name (`id$ClassName`) is useful, if there are multiple composers applied.

```
<window apply="foo.Handle1, foo.Handle2">
    <textbox value="${$Handle1.title}" />
    <textbox value="${$Handle2.name}" />
</window>
```

Specify Name for Composer

If you prefer to name the composer by yourself, you could specify the name in a component attribute called `composerName`. For example,

```
<window apply="MyComposer">
    <custom-attributes composerName="mc"/> <!-- name the composer as mc -->

    <textbox value="${mc.title}" />
</window>
```

Prepare Data for EL Expressions in Composer

It is a common practice to prepare some data in a composer, such that those data are available when rendering the child components. As described above, the composer will be stored as a component attribute that is accessible directly in EL expressions. Thus, you could provide the data easily by declaring a public getter method. For example,

```
public class UsersComposer extends  
org.zkoss.zk.ui.select.SelectorComposer<Window> {  
    public ListModel<User> getUsers() {  
        //return a collection of users  
    }  
}
```

Then, you could access it as follows.

```
<window title="User List" border="normal" apply="foo.UsersComposer">  
    <grid model="${$composer.users}">  
    ...
```

Wire Spring-managed beans

Here is another example that we wire Spring-managed beans with the `@WireVariable` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/select/WireVariable.html#>) annotation.

```
@VariableResolver(org.zkoss.zkplus.spring.DelegatingVariableResolver.class)  
public class UsersComposer extends SelectorComposer<Window> {  
    @WireVariable  
    private List<User> users;  
  
    public ListModel<User> getUsers() {  
        return new ListModelList<User>(users);  
    }  
}
```

where we register a variable resolver called `DelegatingVariableResolver` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkplus/spring/DelegatingVariableResolver.html#>) with the `VariableResolver` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/select/VariableResolver.html#>) annotation. As its name suggests, `DelegatingVariableResolver` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkplus/spring/DelegatingVariableResolver.html#>) will be used to retrieve Spring-managed beans when `@WireVariable` is encountered. For more information, please refer to the `Wire Variables` section.

Notice that the variables will be wired before instantiating the component and its children, so it is OK to access them in the ZUML document, as below.

```
<window title="User List" border="normal" apply="foo.UsersComposer">  
    <grid model="${$composer.users}">  
    ...
```

Composer with More Control

A composer could also handle the exceptions, if any, control the life cycle of rendering, and intercept how a child component is instantiated. It can be done by implementing the corresponding interfaces, ComposerExt (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ComposerExt.html#>) and/or FullComposer (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/FullComposer.html#>).

Initialize Components

If you want to initialize a component's properties with some default values, after ZK creates it, you should override SelectorComposer.doAfterCompose(T) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/select/SelectorComposer.html#doAfterCompose\(T\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/select/SelectorComposer.html#doAfterCompose(T))).

```
public class MyComposer extends SelectorComposer<Grid> {
    public void doAfterCompose(Grid comp) {
        super.doAfterCompose(comp); //wire variables and event listeners
        //initialize wired components here e.g. myLabel.setValue("default
        value")
    }
    ...
}
```

- Line 2: The passed argument, `comp`, is the component that the composer is applied to. In this example, it is the grid. As the name indicates, `doAfterCompose` is called after the grid and all its descendants are instantiated.
- Line 3: Calling `super.doAfterCompose(comp)` first is required to make @Wire and @Listen work.

Exception and Lifecycle Handling with ComposerExt

If you want a composer to handle the exception and/or control the life cycle of rendering, you could also implement ComposerExt (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ComposerExt.html#>). Since SelectorComposer (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/select/SelectorComposer.html#>) already implements this interface, you only need to override the method you care about if you extend from it.

For example, we could handle the exception by overriding ComposerExt.doCatch(java.lang.Throwable) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ComposerExt.html#doCatch\(java.lang.Throwable\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ComposerExt.html#doCatch(java.lang.Throwable))) and/or ComposerExt.doFinally() ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ComposerExt.html#doFinally\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ComposerExt.html#doFinally())).

```
public class MyComposer<T extends Component> extends SelectorComposer<T> {
    public boolean doCatch(Throwable ex) {
        return ignorable(ex); //return true if ex could be ignored
    }
}
```

For involving the life cycle, you could override org.zkoss.zk.ui.Component, org.zkoss.zk.ui.metainfo.ComponentInfo) ComposerExt.doBeforeCompose(org.zkoss.zk.ui.Page, org.zkoss.zk.ui.Component, org.zkoss.zk.ui.metainfo.ComponentInfo) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ComposerExt.html#doBeforeCompose\(org.zkoss.zk.ui.Page,org.zkoss.zk.ui.Component,org.zkoss.zk.ui.metainfo.ComponentInfo\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ComposerExt.html#doBeforeCompose(org.zkoss.zk.ui.Page,org.zkoss.zk.ui.Component,org.zkoss.zk.ui.metainfo.ComponentInfo))) and/or ComposerExt.doBeforeComposeChildren(T) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ComposerExt.html#doBeforeComposeChildren\(T\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ComposerExt.html#doBeforeComposeChildren(T))).

Fine-grained Full Control with FullComposer

In addition to controlling the given component, a composer can monitor the instantiation and exceptions for each child and the descendant component. It is done by implementing FullComposer (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/FullComposer.html#>). SelectorComposer (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/select/SelectorComposer.html#>) does not implement this interface by default. Thus, you have to implement it explicitly.

There is no implementation method needed for this interface. It is like a decorative interface to indicate that it requires the fine-grained full control. In other words, all methods declared in Composer (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Composer.html#>) and ComposerExt (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ComposerExt.html#>) will be invoked one-by-one against each child and the descendant component.

For example, suppose we have a composer implementing both Composer (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Composer.html#>) and FullComposer (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/FullComposer.html#>), and it is assigned as followed

```
<panel apply="foo.MyFullComposer">
    <panelchildren>
        <div>
            <datebox/>
            <textbox/>
        </div>
    </panelchildren>
</panel>
```

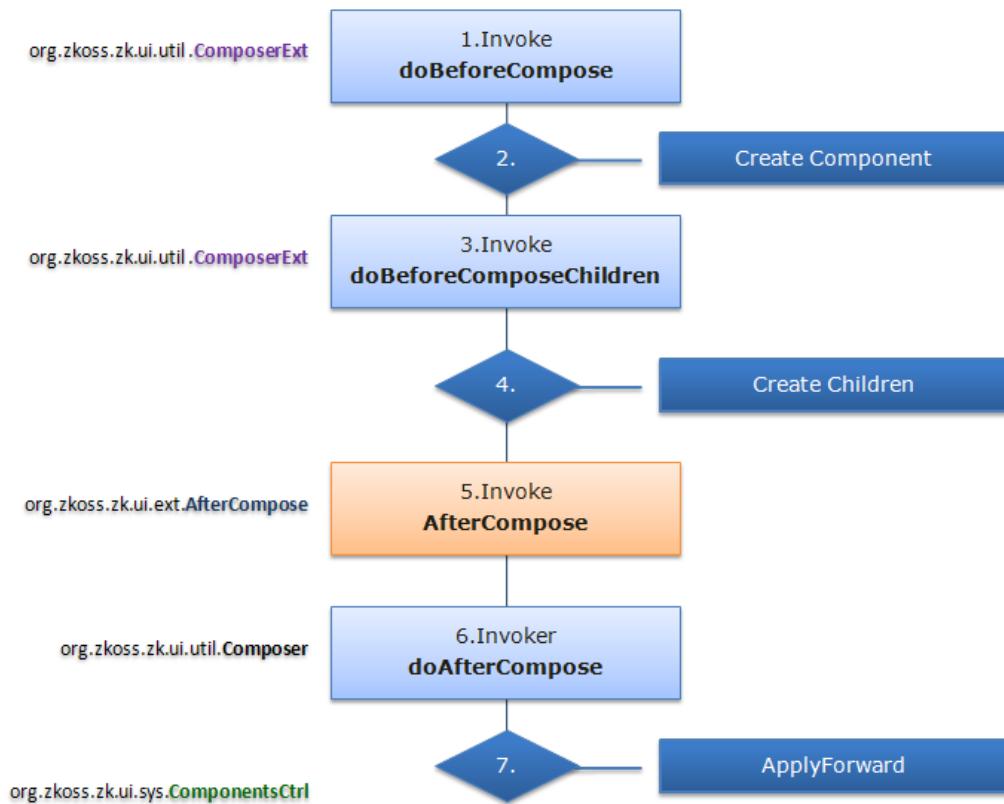
Then, Composer.doAfterCompose(T) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Composer.html#doAfterCompose\(T\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Composer.html#doAfterCompose(T))) will be called for datebox, textbox, div and then panel (in the order of *child-first-parent-last*). If FullComposer (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/FullComposer.html#>) is not implemented, only the panel will be called.

Notice that, because Composer.doAfterCompose(T) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Composer.html#doAfterCompose\(T\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Composer.html#doAfterCompose(T))) will be called for each child, the generic type should be Component (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#>) rather than the component's type to which the composer is applied. For example,

```
public class MyFullComposer extends SelectorComposer<Component> implements
FullComposer {
    ...
}
```

Lifecycle

Here is a lifecycle of the invocation of a composer:



System-level Composer

If you have a composer that shall be invoked for every page, you can register a system-level composer rather than applying it on every page. For example, handling logout logic, or receiving a common event fired by every page.

To register it, specify the composer you implemented in `WEB-INF/zk.xml`:

```

<listener>
    <listener-class>foo.MyComposer</listener-class>
</listener>

```

For more information, please refer to [ZK Configuration Reference/zk.xml](#).

Each time a ZK page, including ZK pages and richlets, is created, ZK will instantiate one instance for each registered system-level composer and then invoke `Composer.doAfterCompose(T)` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Composer.html#doAfterCompose\(T\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Composer.html#doAfterCompose(T))) for each root component. The system-level composer is usually used to process ZK pages after all components are instantiated successfully, such as adding a trademark. If you want to process only certain pages, you can check the request path by calling `Desktop.getRequestPath()` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Desktop.html#getRequestPath\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Desktop.html#getRequestPath())) (the desktop instance can be found through the given component).

If the system-level composer also implements `ComposerExt` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ComposerExt.html#>), it can be used to handle more situations, such as exceptions, like any other composer can do.

If the system-level composer also implements `FullComposer` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/FullComposer.html#>), it will be invoked when each component is created. It provides the finest

grain of control but a wrong implementation might degrade the performance.

Notice that since a new instance of the composer is created for each page, there are no concurrency issues.

Richlet

A system-level composer can implement ComposerExt (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ComposerExt.html#>) to handle exceptions for a richlet, such as `doCatch` and `doFinally`. However, `doBeforeCompose` and `doBeforeComposeChildren` won't be called.

FullComposer (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/FullComposer.html#>) is not applicable to richlets. In other words, system-level composers are called only for root components.

Version History

Version	Date	Content
5.0.8	June, 2011	GenericAutowireComposer (http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/GenericAutowireComposer.html#) and its derives allow developers to specify a custom name by use of a component attribute called <code>composerName</code> .

Wire Components

Wire Components

In a controller that extends SelectorComposer^[3], when you specify a `@Wire` annotation on a field or setter method, the SelectorComposer will automatically find the target component and assign it to the field or pass it into the setter method. It only finds the target component among the applied component and its child components.

Wire with ID selector by Default

You can either specify **component selector syntax**, as the matching criteria for wiring, or leave it empty to wire by **component id (default)**. For example,

ZUL:

```
<window apply="foo.MyComposer">
    <textbox />
    <button id="btn" />
</window>
```

- For this case, `MyComposer` can only wire `<window>` and its child components.

Controller:

```
@Wire
Button btn; // wire to the button with id "btn"
@Wire("window > textbox")
Textbox tb; // wire to the first textbox whose parent is a window
```

CSS3-like Selectors

The string value in @Wire annotation is a **component selector**, which shares an analogous syntax of CSS3 selector. The selector specifies matching criteria against the component tree under the component which applies to this composer.

Given a selector in @Wire annotation, the SelectorComposer will wire a field to the component of **the first match** (in a depth-first-search sense) if the data type of the field is a subtype of Component. Alternatively, if the field type is a subtype of Collection, it will wire to an instance of Collection containing all the matched components.

The syntax elements of selectors are described as the following:

Type

The component type as in ZUML definition, case insensitive.

```
@Wire("button")
Button btn; // wire to the first button.
```

Combinator

Combinator constraints the relative position of components.

```
@Wire("window button")
Button btn0; // wire to the first button who has an ancestor
window
@Wire("window > button") // ">" refers to child
Button btn1; // wire to the first button whose parent is a window
@Wire("window + button") // "+" refers to adjacent sibling (next
sibling)
Button btn2; // wire to the first button whose previous sibling
is a window
@Wire("window ~ button") // "~" refers to general sibling
Button btn3; // wire to the first button who has an older sibling
window
```

You can have any number of levels of combinators:

```
@Wire("window label + button")
Button btn4; // wire to the first button whose previous sibling
is a label with an ancestor window.
```

ID

The component id.

```
@Wire("label#lb")
Label label; // wire to the first label of id "lb" in the same id
space of the root component
@Wire("#btn")
Button btn; // wire to the first component of id "btn", if not a
Button, an exception will be thrown.
```

Unlike CSS3, the id only refers to the component in the same IdSpace of the previous level or root component. For example, given zul

```
<window apply="foo.MyComposer">
    <div>
        <window id="win">
            <div>
                <button id="btn" /><!-- button 1 -->
                <textbox id="tb" /><!-- textbox 1 -->
            </div>
        </window>
        <button id="btn" /><!-- button 2 -->
    </div>
</window>
```

```
@Wire("#btn")
Button btnA; // wire to button 2
@Wire("#win #btn")
Button btnB; // wire to button 1
@Wire("#win + #btn")
Button btnC; // wire to button 2
@Wire("#tb")
Textbox tbA; // fails, as there is no textbox of id "tb"
              // in the id space of the root window (who applies
to the composer).
@Wire("#win #tb")
Textbox tbB; // wire to textbox 1
```

Class

The sclass of component. For example,

```
<window apply="foo.MyComposer">
    <div>
        <button /><!-- button 1 -->
    </div>
    <span sclass="myclass">
        <button /><!-- button 2 -->
    </span>
    <div sclass="myclass">
        <button /><!-- button 3 -->
    </div>
</window>

@Wire(".myclass button")
Button btnA; // wire to button 2
@Wire("div.myclass button")
Button btnB; // wire to button 3
```

Attribute

The attributes on components, which means the value obtained from calling the corresponding getter method on the component.

- Note: `[id="myid"]` does not restrict id space like `#myid` does, so they are **not** equivalent.

```
@Wire("button[label='submit']")
Button btn; // wire to the first button whose getLabel() call
returns "submit"
```

Pseudo Class

A pseudo class is a custom criterion on a component. There are a few default pseudo classes available:

```
@Wire("div:root") // matches only the root component
@Wire("div:first-child") // matches if the component is the first
child among its siblings
@Wire("div:last-child") // matches if the component is the last
child among its siblings
@Wire("div:only-child") // matches if the component is the only
child of its parent
@Wire("div:empty") // matches if the component has no child
@Wire("div:nth-child(3)") // matches if the component is the 3rd
child of its parent
@Wire("div:nth-child(even)") // matches if the component is an
even child of its parent
@Wire("div:nth-last-child(3)") // matches if the component is the
last 3rd child of its parent
@Wire("div:nth-last-child(even)") // matches if the component is
an even child of its parent, counting from the end
```

The `nth-child` and `nth-last-child` pseudo classes parameters can also take a pattern, which follows CSS3 specification [1].

Asterisk

Asterisk simply matches anything. It is more useful when working with combinators:

```
@Wire(" * ")
Component rt; // wire to any component first met, which is the
root.
@Wire("window#win > * > textbox")
Textbox textbox; // wire to the first grandchild textbox of the
window with id "win"
@Wire("window#win + * + textbox")
Textbox textbox; // wire to the second next sibling textbox of
the window with id "win"
```

Multiple Selectors

Multiple selectors separated by commas refer to an OR condition. For example,

```
@Wire("grid, listbox, tree")
MeshElement mesh; // wire to the first grid, listbox or tree
component

@Wire("#win timebox, #win datebox")
InputElement input; // wire to the first timebox or datebox under
window with id "win"
```

Shadow Selectors

Shadow selectors can only be used to select shadow related elements.

One pesudo class **:host** and one pseudo element **::shadow** have been added.

- **:host** select all shadow hosts, which are non-shadow elements, hosting at least one shadow element.
- **:host(selector)** select all shadow hosts matching the additional selector given.
- **::shadow** select all shadow roots, which are shadow elements, hosted by a non-shadow element.

```
<div id="host">

    <apply id="root" dynamicValue="true"><!-- set dynamicValue="true" to avoid being removed after render -->
        <if id="if1" test="@load(vm.showLabel)"><!-- using @load will also prevent this element from being removed -->
            <label id="lb1" value="some text here" />
        </if>
        <if id="if2" test="false"><!-- no dynamicValue or data binding expression, will be removed after render -->
            <label id="lb2" value="some more text here" /><!-- will not render because the if test equals false -->
        </if>
    </apply>
</div>
```

Here are some examples of using shadow selectors with the above zul

```
@Wire(":host") // wire to the div with id "host", as it is the
only shadow host
@Wire(":host(#div2)") // wire to nothing, no shadow host with the
id "div2" exists
@Wire("::shadow") // wire to the apply with id "root", as it is
the only shadow root
@Wire(":host if") // wire to nothing, cannot select from
non-shadow(Div) into shadow(If) without using ::shadow
@Wire(":host::shadow if") // wire to if with the id "if1", as
"if2" will be removed after render, thus cannot be selected
@Wire(":host::shadow if label") // wire to nothing, cannot select
from shadow(If) to non-shadow(Label)
@Wire(":host label") // wire to "lbl", as the host(Div) and the
first label are non-shadow elements
@Wire("#host::shadow#root #if1") // wire to if with the id "if1",
with performance boost
```

Wiring by Method

You can either put the `@Wire` annotation on a field or method. In the latter case, it is equivalent to call the method with the matched component as the parameter. This feature allows a more delicate control on handling auto-wires.

```
@Wire("grid#users")
private void initUserGrid(Grid grid) {
    // ... your own handling
}
```

In the example above, the SelectorComposer will find the grid of id "users" and call `initUserGrid` with the grid as a parameter.

- If the method is static or has wrong signature (more than one parameter), an exception will be thrown.
- Wiring by method **requires a selector** on `@Wire` annotation, otherwise an exception will be thrown.
- If the component is not found, the method is still called, but with `null` value passed in.
- Do not confuse `@Wire` with `@Listen`, while the latter wires to events.

Wiring a Collection

You can also wire **all** matched components to a Collection field or by method if the field is of Collection type or the method takes a Collection as the parameter.

- If the field starts null or uninitialized or wiring by method, SelectorComposer will try to construct an appropriate instance and assign it to the field or pass it to a method call.
- If the field starts with an instance of Collection already, the collection will be cleared and filled with matched components.
- If it wires by method and the selector matches no components, an empty collection will be passed into the method call.

```
@Wire("textbox")
List<Textbox> boxes; // wire to an ArrayList containing all matched
textboxes

@Wire("button")
Set<Button> buttons; // wire to a HashSet containing all matched buttons

@Wire("grid#users row")
List<Row> rows = new LinkedList<Row>(); // the LinkedList will be filled
with matched row components.

@Wire("panel")
public void initPanels(List<Panel> panels) {
    // ...
}
```

Wiring Sequence

While extending from SelectorComposer^[3], the fields and methods with the proper annotations will be wired automatically. Here is the sequence of wiring:

- In Composer.doAfterCompose(T)^[2], it wires components to the fields and methods with the Wire^[3] annotation.
- Before onCreate event of the component which applies to the composer, the SelectorComposer will attempt to wire the **null fields** and **methods** again, for some of the components might have been generated after doAfterCompose() call.

Performance Tips

The selector utility is implemented by a mixed strategy. In a selector sequence, the first few levels with ids specified are handled by Component.getFellow(Component)^[4], and the rest are covered by depth first search (DFS). In brief, the more ids you specify in the **first few levels** of a selector string, the more boost you can obtain in component finding. For example,

```
@Wire("#win #hl > #btn") // fast, as it is entirely handled by
getFellow()

@Wire("window hlayout > button") // slower, entirely handled by
DFS

@Wire("#win hlayout > button") // first level is handled by
getFellow(), other handled by DFS

@Wire("window #hl > #btn") // slower, as the first level has no
id, all levels are handled by DFS
```

- Note: specifying id via attribute (for instance, [id='myid']) does not lead to the same performance boost.

In the case of multiple selectors, only the first few **identical levels with ids** enjoy the performance gain.

```
@Wire("#win #hl > button, #win #hl > toolbarbutton")
// the first two levels have boost

@Wire("#win #hl > btn, #win #hl > #toolbtn")
// the first two levels have boost

@Wire("#win + #hl > btn, #win #hl > #btn")
// only the first level has boost, as they differ in the first
combinator

@Wire("#win hlayout > btn, #win hlayout > #toolbtn")
// only the first level has boost, as the second level has no id
specified
```

In brief, it is recommended to specify id in selector when you have a large component tree. If possible, you can specify id on all levels to maximize the performance gain from the algorithm.

References

- [1] <http://www.w3.org/TR/css3-selectors/#nth-child-pseudo>
- [2] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Composer.html#doAfterCompose\(T\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Composer.html#doAfterCompose(T))
- [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/select/annotation/Wire.html#>
- [4] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#getFellow\(Component\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#getFellow(Component))

Wire Variables

Wire Variables

SelectorComposer^[3] not only wires UI components, but also wires beans from implicit objects and registered variable resolvers.

Wire from Implicit Objects

Wiring from an implicit object is equivalent to calling java.lang.String Components.getImplicit(org.zkoss.zk.ui.Page, java.lang.String)^[1], by the name specified on @WireVariable. If the name is absent and the field or method parameter is of type Execution^[1], Page^[8], Desktop^[9], Session^[2], or WebApp^[3], it still will be wired to the correct implicit object. However, in other cases, an exception will be thrown.

```
public class FooComposer extends SelectorComposer<Window> {

    @WireVariable
    private Page _page;

    @WireVariable
    private Desktop _desktop;

    @WireVariable
    private Session _sess;

    @WireVariable
    private WebApp _wapp;

    @WireVariable("desktopScope")
    private Map<String, Object> _desktopScope;

}
```

Wire from Variable Resolver

There are two approaches to register a variable resolver: the VariableResolver^[4] annotation or the variable-resolver directive. Here is the example of registering variable resolvers with annotations.

```
@VariableResolver({foo1.MyResolver.class, foo2.AnotherResolver.class})
public class FooComposer extends SelectorComposer<Grid> {
    ...
}
```

To have SelectorComposer^[3] wire a variable, you have to annotate it with the WireVariable^[5] annotation. For example,

```
@VariableResolver({foo1.MyResolver.class, foo2.AnotherResolver.class})
public class FooComposer extends SelectorComposer<Grid> {
    @WireVariable
    Department department;
    @WireVariable
    public void setManagers(Collection<Manager> managers) {
        //...
    }
}
```

Wire Spring-managed Beans

If you'd like SelectorComposer^[3] to wire the Spring-managed beans, you can register the Spring variable resolver, DelegatingVariableResolver^[9] with @VariableResolver. Then, you can annotate @WireVariable to wire a Spring-managed bean. It's wired according to its variable name as the bean's *id*. For example,

```
@VariableResolver(org.zkoss.zkplus.spring.DelegatingVariableResolver.class)
public class PasswordSetter extends SelectorComposer<Window> {
    @WireVariable
    private User user;
    @Wire
    private Textbox password; //wired automatically if there is a
    textbox named password

    @Listen("onClick=#submit")
    public void submit() {
        user.setPassword(password.getValue());
    }
}
```

DelegatingVariableResolver^[9] is a variable resolver used to retrieve the Spring-managed bean, so the variable will be retrieved and instantiated by Spring.

Notice that the variables are wired before instantiating the component and its children, so you can use them in EL expressions. For example, assume we have a composer as follows.

```
@VariableResolver(org.zkoss.zkplus.spring.DelegatingVariableResolver.class)
public class UsersComposer extends SelectorComposer<Window> {
    @WireVariable
    private List<User> users;

    public ListModel<User> getUsers() {
        return new ListModelList<User>(users);
    }
}
```

Then, you could reference to getUsers() in the ZUML document. For example,

```
<window apply="UsersComposer">
  <grid model="\${$composer.users}">
  ...

```

where `$composer` is a built-in variable referring to the composer. For more information, please refer to the Composer section.

-
- [1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Components.html#getImplicit\(org.zkoss.zk.ui.Page,%20java.util.Map\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Components.html#getImplicit(org.zkoss.zk.ui.Page,%20java.util.Map))
 - [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Session.html#>
 - [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/WebApp.html#>
 - [4] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/select/annotation/VariableResolver.html#>
 - [5] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/select/annotation/WireVariable.html#>

Warning: Not a good idea to have Spring managing the composer

There is a tendency to make the composer a Spring-managed bean. For example, assume we have a composer called `passwordSetter` and managed by Spring, then we might do as follows.

```
<?variable-resolver class="org.zkoss.zkplus.spring.DelegatingVariableResolver"?>
<window apply="\${passwordSetter}">
...
@Component
public class PasswordSetter extends SelectorComposer {
  @Autowired User user;
...

```

Unfortunately, this approach is error-prone. The reason is that none of Spring's scopes matches correctly with the lifecycle of the composers. For example, if the Session scope is used, it will cause errors when the user opens two browser windows to visit the same page. In this case, the same composer will be used to serve all desktops in the given session, and it is wrong.

The Prototype scope is a better choice since a new instance is instantiated for each request. However, it also implies another new instance will be instantiated if the Spring variable resolver is called to resolve the same name again in the later requests. It is unlikely, but it might be triggered implicitly and hard to debug. For example, it happens if some of your code evaluates an EL expression that references the composer's name, when an event is received.

ZK Spring (<http://www.zkoss.org/product/zkspring>) is recommended if you want to use Spring intensively. It extends Spring to provide the scopes matching ZK lifecycle, such as the IdSpace and Component scopes. Please refer to ZK Spring Essentials for more detailed information.

Wire other variables in GenericForwardComposer based composers

Composers extending GenericAutowireComposer (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/GenericAutowireComposer.html#>) such as GenericForwardComposer (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/GenericForwardComposer.html#>) will automatically wire variables based on convention during `doAfterCompose(Component comp)`.

This method supports autowiring based on naming convention. You don't need to specify annotations explicitly, but it is error-prone if it is used improperly.

This wiring by convention will wire variables based on their name in composer, if a bean registered with the same name can be found in the following locations:

- If enabled: ZScript variable of same name;

- If enabled: Xel variable of same name
- Attribute of component holding the composer declaration
- Attribute of component's ancestor components
- Attribute of component's page
- Attribute of component's desktop
- Attribute of component's session
- Attribute of component's webapp

Searching in zscript and xel variables can be enabled with library properties:

enable zscript variable wiring

enable xel variable wiring

Wire CDI-managed Beans

The approach to work with CDI is similar to the approach for Spring, except the variable resolver for CDI is DelegatingVariableResolver (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkplus/cdi/DelegatingVariableResolver.html#>).

Wiring Sequence

When extending from SelectorComposer (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/select/SelectorComposer.html#>), the fields and methods with the proper annotations will be wired automatically. Here is the sequence of wiring:

- In org.zkoss.zk.ui.Component, org.zkoss.zk.ui.metainfo.ComponentInfo)

ComposerExt.doBeforeCompose(org.zkoss.zk.ui.Page, org.zkoss.zk.ui.Component, org.zkoss.zk.ui.metainfo.ComponentInfo) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ComposerExt.html#doBeforeCompose\(org.zkoss.zk.ui.Page,%20org.zkoss.zk.ui.Component,%20org.zkoss.zk.ui.metainfo.ComponentInfo\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ComposerExt.html#doBeforeCompose(org.zkoss.zk.ui.Page,%20org.zkoss.zk.ui.Component,%20org.zkoss.zk.ui.metainfo.ComponentInfo))), it wires variables to the fields and methods annotated with the WireVariable (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/select/annotation/WireVariable.html#>) annotation. Here is the sequence of how it looks for the variable:

 1. First, it will look for the variable resolver defined in the ZUML document (by use of Page.addVariableResolver(org.zkoss.xel.VariableResolver) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Page.html#addVariableResolver\(org.zkoss.xel.VariableResolver\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Page.html#addVariableResolver(org.zkoss.xel.VariableResolver)))).
 2. Second, it looks for the variable resolver annotated at the class with the VariableResolver (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/select/annotation/VariableResolver.html#>) annotation.
 3. If none is found, it looks for the implicit objects, such as session and page.

Version History

Version	Date	Content
6.0.0	February 2012	@WireVariable was introduced.

Wire Event Listeners

Wire Event Listeners

To wire an event listener, you need to declare a method with Listen^[3] annotation. The method should be public, with return type void, and have either no parameter or one parameter of the specific event type (corresponding to the event being listened to). The parameter of @Listen should be pairs of an event name and a selector, separated by a semicolon.

ZK will look for the target component specified by the selector **in the same ID space**.

For example,

```
@Listen("onClick = #btn0")
public void submit(MouseEvent event) {
    // called when onClick is received on the component of id btn0.
}

@Listen("onSelect = #listbox0")
public void select(SelectEvent event) {
    // called when onSelect is received on the component of id
listbox0.
}
```

Event Listener Parameter

There are three ways to declare the method signature of the event listener:

1. No parameter
2. One parameter of the corresponding event type
3. One parameter of a super class of the corresponding event type

For example,

```
@Listen("onChange = textbox#input0")
public void change() {
    // called when onChange is received on the textbox of id input0.
}

@Listen("onChange = textbox#input1")
public void change(InputEvent event) {
    // called when onChange is received on the textbox of id input1.
}

@Listen("onChange = textbox#input2")
public void change(Event event) {
    // called when onChange is received on the textbox of id input2.
}
```

Multiple Targets

If the selector matches multiple components, the event listener will be wired to **all** matched components. In such case, if you need to know which component receives the event, you can retrieve it from `Event#getTarget()`.

For example,

```
@Listen("onClick = grid#myGrid > rows > row")
public void click(MouseEvent event) {
    // called when onClick is received on any Row directly under the
    Grid of id myGrid
}

@Listen("onClick = #btn0, #btn1, #btn2")
public void click(MouseEvent event) {
    // called when onClick is received on components of id #btn0,
    #btn1, or #btn2
}
```

Multiple Event Types

By separating multiple pairs of event names and selectors by **semicolons**, you can wire different types of events to a single method.

For example,

```
@Listen("onClick = button#submit; onOK = textbox#password")
public void submit(Event event) {
    // called when onClick is received on #submit, or onOK (Enter key
    pressed) is received on #password
}
```

Event Listener Priority

The calling order of an event listener can be declared by specifying a number after the event name. The event listeners will be executed in the order from the largest to smallest of the declared priority number. Listeners without a priority number will be set to 0 automatically.

For example,

```
@Listen("onClick(1) = #btn0")
public void submit1(MouseEvent event) {
    // called before submit2 method when onClick is received on the
    component of id btn0.
}

@Listen("onClick = #btn0")
public void submit2(MouseEvent event) {
    // the priority of this listener will be set to 0 by default,
    and will be called after submit1 and before submit2 method when onClick
    is received on the component of id btn0.
}

@Listen("onClick(-1) = #btn0")
public void select3(SelectEvent event) {
```

```
// called after submit2 method when onClick is received on the
component of id btn0.
}
```

Version History

Version	Date	Content
6.0.0	February 2012	@Listen was introduced.

Subscribe to EventQueues

Subscribe by @Subscribe

- Available for ZK:
- CE** **PE** **EE**

A method (as if in an EventListener) in SelectorComposer^[1] can subscribe to an EventQueue by @Subscribe^[2]. For example,

```
// in sender composer
public void publish() {
    EventQueue<Event> eq = EventQueues.lookup("queue1", EventQueues.DESKTOP,
true);
    eq.publish(new Event("onMyEvent", component, data));
}
```

```
// in receiver composer
@Subscribe("queue1")
public void receive(Event event) {
    // this method will be called when EventQueue "queue1" of Desktop
scope is published
    Object data = event.getData();
    Component target = event.getTarget();
}
```

- Notice the queue name should match.
- ZK executes both methods in a servlet thread, so if they execute a time-consuming operation, they will block users.

In the example above, when you publish an event in the EventQueue, the subscribed method will be called. This is a useful mechanism to communicate among composers. See also EventQueue^[4].

EventQueue Scope

You can subscribe to EventQueue of different scopes.

```
@Subscribe(value = "queue2", scope = EventQueues.SESSION)
public void method2(Event event) {
    // this method will be called when EventQueue "queue2" of Session
    scope is published
}
public void publish() {
    EventQueue<Event> eq = EventQueues.lookup("queue2", EventQueues.SESSION,
    true);
    eq.publish(new Event("onMyEvent", component, data));
}
```

Available scopes are: Desktop, Group, Session, Application. Note that Group scope requires ZK EE. See also EventQueues^[1].

Event Name

You can also listen to a specified event name.

```
@Subscribe(value = "queue2", eventName = "event1")
public void method2(Event event) {
    // this method will be called when EventQueue "queue2" of Session
    scope is published
}
public void publish() {
    EventQueue<Event> eq = EventQueues.lookup("queue2", EventQueues.DESKTOP,
    true);
    eq.publish(new Event("event1", component, data));
}
```

Subscriber Method Parameter

The method which subscribes to the EventQueue takes either no parameter or one parameter of a type Event.

```
@Subscribe("queue3")
public void method3() { // the event parameter can be omitted
    // ...
}
```

ZK automatically maps event data into the method parameters in order.

```
@Subscribe("queue3")
public void method3(int i, String s) {
    // i will be 100, s will be "EventData"
    // ...
}

public void publish() {
```

```

EventQueue<Event> eq = EventQueues.lookup("queue3", EventQueues.DESKTOP,
true);
eq.publish(new Event("event1", component, new Object[]{100,
"eventData"}));
}

```

If you put the event at the first one, it also works well.

```

@Subscribe("queue3")
public void method3(Event event, int i, String s) {
    // ...
}

```

To recap, we now have four ways to use a parameter:

- method()
- method(Event event)
- method(Event event, int d1, String d2,)
- method(int d1, String d2, ...)

Auto-Unsubscribed

@Subscribe will unsubscribe the subscribed event-queue automatically when the applied component (or its ancestor) of a composer is detached.

Version History

Version	Date	Content
6.0.1	April 2012	@Subscribe was introduced.
7.0.3	June 2014	ZK-2076 [3] Enhance Subscribe annotation to map java method by the event name and the parameter type in order

References

- [1] <https://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/select/SelectorComposer.html>
- [2] <https://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkmax/ui/select/annotation/Subscribe.html>
- [3] <http://tracker.zkoss.org/browse/ZK-2076>

Model

The *model* is the data an application handles. Depending on the application requirement, it could be anything as long as your controller knows it. Typical objects are POJOs, beans, Spring-managed beans, and DAOs. Examples of manipulating the model in the controller were discussed in the previous sections.

In this section and subsections, we will focus on the model that ZK components support directly without custom glue logic. For example, implementing ListModel^[7] to control the display of Listbox^[3] and Grid^[3], and ChartModel^[1] to control Chart^[2].

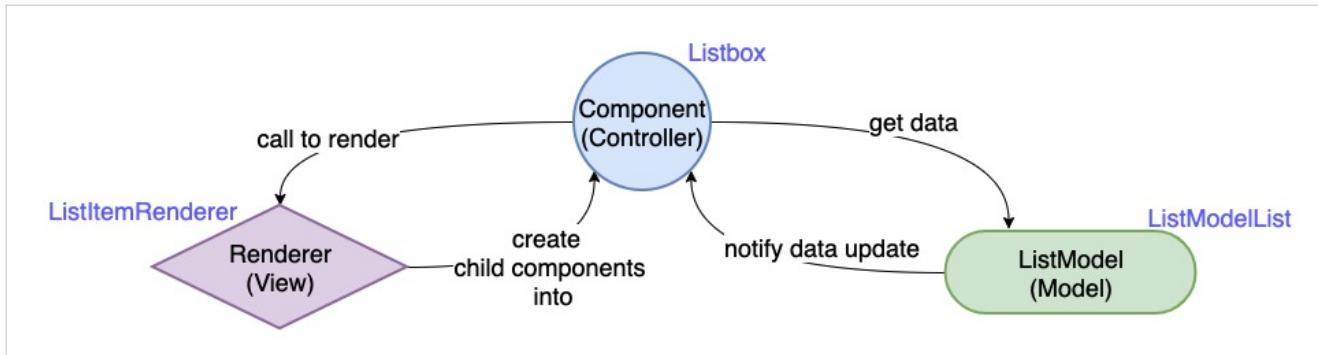
In addition to implementing these models, you could use one of the predefined implementations such as SimpleListModel^[3] and SimplePieModel^[4]. For detailed description, please refer to the following sections.

Model-Driven Rendering

Those ZK components (e.g. Listbox, Grid, Tree) that allow you to set `model` properties support model-driven rendering, which means they create (render) their child components upon a model object (e.g. ListModelList). Hence, you can assign a ListModelList to a Listbox, and it will render its Listitem upon the ListModelList with its renderer.

In this case, you have to control the component's rendering by **manipulating its model objects**. If you want to remove a Listitem, don't remove the Listitem object itself. You should remove the corresponding object in the ListModelList (`remove()`). Then Listbox will remove the Listitem accordingly. The same rule applies to adding, clearing, and replacing.

From one component's perspective, it also follows the MVC pattern:



- **Component (Controller)**
 - get data objects from a Model
 - as a data change listener of Model (by invoking `ListModel.addListDataListener(org.zkoss.zul.event.ListDataListener)`^[5])
 - handle events from the client e.g. update the selection upon user clicking
- **ListModel (Model)**
 - it stores your domain objects (the data could be Car, Person)
 - notify the component for data update when you call its method e.g. `add()`, `remove()`, `clear()`
 - ZK provides various default model implementations for various components, please refer to ListModel^[6], TreeModel^[7], GroupsModel^[8]
- **Renderer (View)**
 - render/create child components and insert them into their parent component according to the request from the Component

- each component has its own built-in renderer

How to Assign Model to UI

Depending on the requirements, there are a few ways to assign a model to a UI component.

Use Composer to Assign Model

A typical way is to use a composer to assign the model. For example, assume the UI component is a grid and we have a method called `getFooModel` returning the data to show on the grid, then we could implement a composer, say `foo.FooComposer` as follows:

```
public class FooComposer implements Composer {  
    public void doAfterCompose(Component comp) throws Exception {  
        ((Grid) comp).setModel(getFooModel());  
    }  
}
```

Then, you could assign it in ZUML as follows:

```
<grid apply="foo.FooComposer">  
...
```

Use Data Binding

If you are using data binding, you can have the data binder to assign the model for you. For example, assume that you have a `ListModelList persons`, then:

```
<listbox model="@init(vm.persons)">
```

Use EL Expressions

EL is another common way to assign the model. For example, assume you have a variable resolver called `foo.FooVariableResolver` implementing `VariableResolver`^[2] as follows.

```
public class FooVariableResolver implements VariableResolver {  
    public Object resolveVariable(String name) {  
        if ("persons".equals(name)) //found  
            return getPersons(); //assume this method returns an  
instance of ListModel  
            //... you might support more other variables  
        return null; //not found  
    }  
}
```

Then, you could specify it in ZUML as follows:

```
<?variable-resolver class="foo.FooVariableResolver"?>  
  
<listbox model="${persons}">  
...
```

The other approach is to use the function mapper. For example, assume you have an implementation called `foo.CustomerListModel`, then you could use it to drive a listbox as follows.

```
<?taglib uri="http://www.zkoss.org/dsp/web/core" prefix="c" ?>
<listbox model="${c:new('foo.CustomerListModel')}"/>
```

Use zscript

If you are building a prototype, you could use zscript to assign the model directly. For example,

```
<zk>
  <zscript>
    ListModel infos = new ListModelArray(
      new String[][] {
        {"Apple", "10kg"},
        {"Orange", "20kg"},
        {"Mango", "12kg"}
      });
  </zscript>
  <listbox model="${infos}" />
</zk>
```

Notice that, since the performance of zscript is not good and the mix of Java code in ZUML is not easy to maintain, it is suggested **not** to use this approach in a production system. Please refer to Performance Tips for more information.

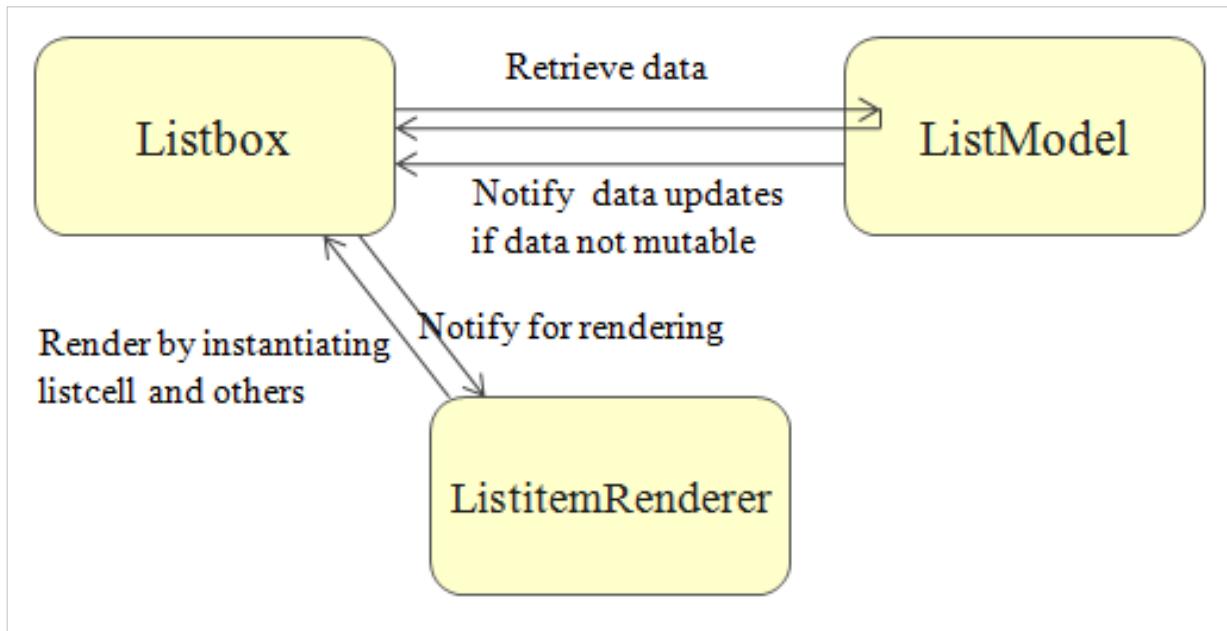
References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ChartModel.html#>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Chart.html#>
- [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/SimpleListModel.html#>
- [4] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/SimplePieModel.html#>
- [5] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ListModel.html#addListDataListener\(org.zkoss.zul.event.ListDataListener\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ListModel.html#addListDataListener(org.zkoss.zul.event.ListDataListener))
- [6] <https://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ListModel.html>
- [7] <https://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/TreeModel.html>
- [8] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/GroupsModel.html>

List Model

Listbox^[3], Grid^[3], and Tabbox^[1] allow developers to separate the view and the model by implementing ListModel^[7]. Once the model is assigned (with Listbox.setModel(org.zkoss.zul.ListModel)^[2]), the display of the listbox is controlled by the model, and an optional renderer. The model is used to provide data, while the renderer is used to provide the custom look. By default, the data is shown as a single-column grid/listbox. If it is not what you want, please refer to the View section for writing a custom renderer.

Model-driven Rendering



As shown, the listbox retrieves elements from the specified model^[3], and then invokes the renderer, if specified, to compose the listitem for the element.

The retrieval of elements is done by invoking ListModel.getSize()^[4] and ListModel.getElementAt(int)^[5].

The listbox will register itself as a data listener to the list model by invoking ListModel.addListDataListener(org.zkoss.zul.event.ListDataListener)^[5]. Thus, if the list model is not mutable, the implementation has to notify all the registered data listeners. It is generally suggested to extend from AbstractListModel^[6], or use any of the default implementations, which provide a set of utilities for handling data listeners transparently. We will talk about it later in #Notify for Data Updates.

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

[2] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Listbox.html#setModel\(org.zkoss.zul.ListModel\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Listbox.html#setModel(org.zkoss.zul.ListModel))

[3] The listbox is smart enough to read the elements that are visible at the client, such as the elements for the active page. It is called *Live Data or Render on Demand*.

[4] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ListModel.html# getSize\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ListModel.html# getSize())

[5] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ListModel.html# getElementAt\(int\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ListModel.html# getElementAt(int))

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

Default ListModel Implementation

In most cases, you can use ZK default implementation of ListModel (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ListModel.html#>) as the model without implementing by yourselves:

- ListModelList (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ListModelList.html#>)
- ListModelArray (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ListModelArray.html#>)
- ListModelSet (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ListModelSet.html#>)
- ListModelMap (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ListModelMap.html#>)

For example,

```
void setModel(List data) {
    listbox.setModel(new ListModelList(data));
}
```

Benefit: Optimizing Rendering at the Client-side

When you call a method of ListModel, e.g. add(), remove(), set(), it will automatically notify its associated component to render the differential items instead of re-rendering all items.

Load All Data into a ListModel

If the amount of your data is small, you could load them all into a list, map, set or array. Then, you could use one of the default implementations as described above.

Alternatively, you could load all data when ListModel.getSize() ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ListModel.html# getSize\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ListModel.html# getSize())) is called. For example,

```
public class FooModel extends AbstractListModel {
    private List _data;
    public int getSize() {
        //load all data into _data
        return _data.size();
    }
    public Object getElementAt(int index) {
        return _data.get(index);
    }
}
```

Load Partial Data into a ListModel

If the data amount is huge, it is not a good idea to load all of them at once. Rather, you shall load only the required subset. On the other hand, it is generally not a good idea to load single elements when ListModel.getElementAt(int) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ListModel.html# getElementAt\(int\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ListModel.html# getElementAt(int))) is called since the overhead loading from the database is significant.

Thus, it is suggested to use SQL LIMIT or a similar feature to load only a subset of data. For example, if the total number of visible elements is about 30, you could load 30 (or more, say 60, depending on performance or memory is more important to you). If an element is not loaded, you have to discard the previously loaded data if any. If the next invocation of ListModel.getElementAt(int) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ListModel.html# getElementAt\(int\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ListModel.html# getElementAt(int))) is in the subset, we could return it immediately. Here is the pseudo code:

```

public class FooModel extends AbstractListModel {
    public List _subset;
    public int _startAt;

    public Object getElementAt(int index) {
        if (index >= _startAt && _subset != null && index - _startAt < _subset.size())
            return _subset.get(index - _startAt); //cache hit
        _subset = new LinkedList(); //drop _subset, and load a subset
        of data, say, 60, to _subset
        ...
    }
}

```

For more realistic examples, please refer to example project (<https://github.com/zkoss/zkbooks/blob/master/developersreference/developersreference/src/main/webapp/mvc/model/listModel.zul>).

Notify for Data Updates

If you build your own `ListModel` implementation, when the data in the model is changed, the implementation must notify all the data listeners that are registered by `ListModel.addDataListener(org.zkoss.zul.event.ListDataListener)` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ListModel.html#addListDataListener\(org.zkoss.zul.event.ListDataListener\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ListModel.html#addListDataListener(org.zkoss.zul.event.ListDataListener))). It can be done by invoking `int, int) AbstractListModel.fireEvent(int, int, int)` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/AbstractListModel.html#fireEvent\(int, int, int\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/AbstractListModel.html#fireEvent(int, int, int))) if your implementation is extended from `AbstractListModel` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/AbstractListModel.html#>) or derived.

Notice that if you use one of the default implementations, such as `ListModelList` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ListModelList.html#>), you don't need to worry about it. The notification is handled transparently.

For example, (pseudo code)

```

public void removeRange(int fromIndex, int toIndex) {
    removeElements(fromIndex, toIndex); //remove elements from
    fromIndex (inclusive) to toIndex (exclusive)
    fireEvent(ListDataEvent.INTERVAL_REMOVED, fromIndex, index - 1);
}

public void add(int index, Object element) {
    addElements(index, element); //add an element at index
    fireEvent(ListDataEvent.INTERVAL_ADDED, index, index);
}

public void set(int index, Object element) {
    setElement(index, element); //change the element at index
    fireEvent(ListDataEvent.CONTENT_CHANGED, index, index);
}

```

Once a model is assigned to a component, the component will register itself as a data listener such that any changes can be updated to UI.

Notice that you shall not update the component (such as listbox) directly. Rather, you shall update the model and then the model shall fire an event to notify the components to render accordingly.

Selection

Select through ListModel

It's important that, once you assign a `ListModel` to a `Listbox`, you shall not manipulate a `Listitem` and/or change the selection through a `Listbox` API directly. Rather, the application shall **add, remove and select data items via `ListModel` API**, and let the model notify the component what has been changed. For example, here are the most 2 commonly used:

- `addToSelection()` (<https://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/AbstractListModel.html#addToSelection-E->)
- `removeFromSelection()` (<https://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/AbstractListModel.html#removeFromSelection-java.lang.Object->)

Get Selection

You can get selected objects by `getSelection()` (<https://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/AbstractListModel.html#getSelection-->)

```
private ListModelList<Locale> listModel = new
ListModelList<>(Locale.getAvailableLocales());
...
listModel.getSelection().iterator(); //iterate it to get all selected
Locale objects
```

Selection Control

With the multiple selection function in a data model, you have to implement a class for the `SelectionControl` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/SelectionControl.html#>) to tell the data model which items are selectable and what it will perform a "select all" function with. The following implementation extends `DefaultSelectionControl` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/AbstractListModel/DefaultSelectionControl.html#>) and serves as a simple example to change "selectable" items.

Please note that if your data model is much larger, you may implement it on your own to get rid of the performance impact.

```
model.setSelectionControl(new
AbstractListModel.DefaultSelectionControl(model) {
    public boolean isSelectable(Object e) {
        int i = model.indexOf(e);
        return i % 2 == 0;
    }
});
```

Custom ListModel Supports Selection

Interface: [Selectable](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext>Selectable.html#)
 Implementation: Implemented by [AbstractListModel](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/AbstractListModel.html#)

If your data model also provides the collection of selected elements, you shall also implement Selectable (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext>Selectable.html#>). When using with a component supporting the selection (such as Listbox (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul>Listbox.html#>)), the component will invoke Selectable.isSelected(E) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext>Selectable.html#isSelected\(E\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext>Selectable.html#isSelected(E))) to display the selected elements correctly. In addition, if the end user selects or deselects an item, Selectable.addSelection(E) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext>Selectable.html#addSelection\(E\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext>Selectable.html#addSelection(E))) and Selectable.removeSelection(java.lang.Object) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext>Selectable.html#removeSelection\(java.lang.Object\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext>Selectable.html#removeSelection(java.lang.Object))) will be called by the component to notify the model that the selection is changed. Then, you can update the selection into the persistent layer (such as database) if necessary.

On the other hand, when the model detects the selection is changed (such as Selectable.addSelection(E) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext>Selectable.html#addSelection\(E\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext>Selectable.html#addSelection(E)))) is called), it has to fire the event, such as ListDataEvent.SELECTION_CHANGED (http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/event>ListDataEvent.html#SELECTION_CHANGED) to notify the component. It will cause the component to correct the selection^[1].

All default implementations, including AbstractListModel (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/AbstractListModel.html#>), implement Selectable (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext>Selectable.html#>). Thus, your implementation generally doesn't need to handle the selection if it extends one of these classes.

[1] Don't worry. The component is smart enough to prevent the dead loop, even though the component invokes addSelection to notify the model while the model fires the event to notify the component.

Sorting

Interface: [Sortable.html#">Sortable](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext)
 Implementation: You have to implement it explicitly

To support sorting, the model must implement Sortable (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext>) too. Thus, when the end user clicks the header to request sorting, boolean) Sortable.sort(java.util.Comparator, boolean) (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext>) will be called.

For example, (pseudo code)

```
public class FooModel extends AbstractListModel implements Sortable {
    public void sort(Comparator cmpr, final boolean ascending) {
        sortData(cmpr); //sort your data here
        fireEvent(ListDataEvent.CONTENT_CHANGED, -1, -1); //ask
component to reload all
    }
    ...
}
```

Notice that the ascending parameter is used only for reference and you usually don't need it, since the cmpr is already a comparator capable to sort in the order specified in the ascending parameter.

Version History

Version	Date	Content
6.0.0	February 2012	All selection states are maintained in the list model. And, the application shall <i>not</i> access the component for the selection. Rather, the application shall invoke Selectable (Selectable.html#">http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext>Selectable.html#) for retrieving or changing the selection.
6.0.0	February 2012	Sortable (http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/Sortable.html#) was introduced and replaced ListModelExt.

Groups Model

- Available for ZK:

CE PE EE

Here we describe how to implement a groups model (GroupsModel^[1]). For the concept of component, model and render, please refer to the Model-driven Display section.

A groups model is used to drive components that support groups of data. The groups of data is a two-level tree of data: a list of grouped data and each grouped data is a list of elements to display. Here is a live demo^[2]. Currently, both Listbox^[3] and Grid^[3] support a list of grouped data.

Instead of implementing GroupsModel^[1], it is suggested to extend from AbstractGroupsModel^[3], or to use one of the default implementations as following:

	SimpleGroupsModel ^[4]	GroupsModelArray ^[5]
Usage	The grouping is immutable , i.e., re-grouping is not allowed	Grouping is based on a comparator (java.util.Comparator)
Constructor	The data must be grouped, i.e., data[0] is the first group, data[1] the second, etc.	The data is <i>not</i> grouped, i.e., data[0] is the first element. The constructor requires a comparator that will be used to group them.
Version	Since 3.5.0	Since 5.0.5; For 5.0.4 or prior, please use org.zkoss.zul.ArrayGroupsModel ^[6] (the same).

Example: Immutable Grouping Data

If your data is already grouped and the grouping won't be changed, then you could use SimpleGroupsModel^[4] as follows:

```
<zk>
<zscript>
String[][] datas = new String[][] {
    new String[] { //group 1
        // Today
        "RE: Bandbox Autocomplete Problem",
        "RE: It's not possible to navigate a listbox' item",
        "RE: FileUpload"
    },
    new String[] { //group 2
        // Yesterday
    }
}
```

```

        "RE: Opening more than one new browser window",
        "RE: SelectedItemConverter Question"
    },
    new String[] { //group 3
        "RE: Times_Series Chart help",
        "RE: SelectedItemConverter Question"
    }
};

GroupsModel model = new SimpleGroupsModel(datas,
    new String[]{"Date: Today", "Date: Yesterday", "Date: Last
Week"});

//the 2nd argument is a list of group head
</zscript>
<grid model="#${model}">
    <columns sizable="true">
        <column label="Subject"/>
    </columns>
</grid>
</zk>
```

Then, the result

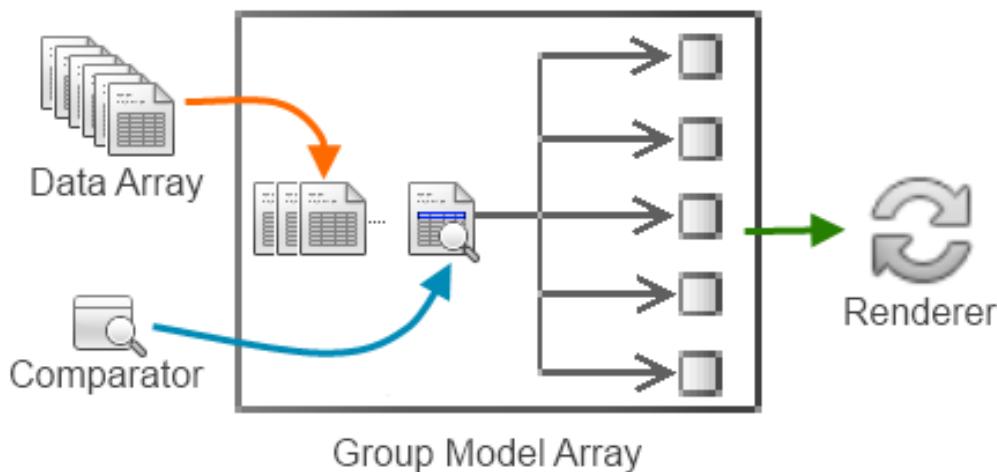
Subject
▲ Date: Today
RE: Bandbox Autocomplete Problem
RE: It's not possible to navigate a listbox' ite
RE: FileUpload
▲ Date: Yesterday
RE: Opening more than one new browser window
RE: SelectedItemConverter Question
▲ Date: Last Week
RE: Times_Series Chart help
RE: SelectedItemConverter Question

Sorting and Regrouping

If your groups model allows the end user to sort and/or to re-group (i.e., grouping data based on different criteria), you have to implement GroupsSortableModel^[7] too. Then, GroupsSortableModel.group(java.util.Comparator,boolean,int)^[8] will be called if the user requests to re-group the data based on a particular column. And, GroupsSortableModel.sort(java.util.Comparator,boolean,int)^[9] will be called if the user requests to sort the data based on a particular column.

GroupsModelArray^[5] supports both sorting and re-grouping as described below:

- Sorting: GroupsModelArray^[5] sorts each group separately by using the specified comparator (java.util.Comparator).
- Re-grouping: GroupsModelArray^[5] re-groups by assuming two data belong to the same group if the compared result is the same (i.e., the given java.util.Comparator returns 0).
- For better control, you could implement GroupComparator^[10], and pass an instance to, say, Column.setSortAscending(java.util.Comparator)^[11] and Column.setSortDescending(java.util.Comparator)^[12].



Example: Grouping Tabular Data

Suppose you have the data in a two-dimensional array (see below), and you want to allow the user to group them based on a field selected by the user (such as food's name or food's calories).

Category	Name	Top Nutrient	% of Daily	Calories
▲ 10			Group	
Vegetables	Eggplant	Dietary Fiber	Sort Ascending	
▲ 21			Sort Descending	
Vegetables	Onions	Chromium	Category	
▲ 24			Name	
Grains	Corn	Vitamin B1	Top Nutrients	
Fruits	Strawberries	Vitamin C	% of Daily	
Fruits	Watermelon	Vitamin C	Calories	
▲ 26				

```
//Data
Object[][] _foods = new Object[][][] { //Note: the order does not matter
    new Object[] { "Vegetables", "Asparagus", "Vitamin K", 115, 43},
    new Object[] { "Vegetables", "Beets", "Folate", 33, 74},
    new Object[] { "Vegetables", "Bell peppers", "Vitamin C", 291, 24},
    new Object[] { "Vegetables", "Cauliflower", "Vitamin C", 92, 28},
```

```

    new Object[] { "Vegetables", "Eggplant", "Dietary Fiber", 10, 27},
    new Object[] { "Vegetables", "Onions", "Chromium", 21, 60},
    new Object[] { "Vegetables", "Potatoes", "Vitamin C", 26, 132},
    new Object[] { "Vegetables", "Spinach", "Vitamin K", 1110, 41},
    new Object[] { "Vegetables", "Tomatoes", "Vitamin C", 57, 37},
    new Object[] { "Seafood", "Salmon", "Tryptophan", 103, 261},
    new Object[] { "Seafood", "Shrimp", "Tryptophan", 103, 112},
    new Object[] { "Seafood", "Scallops", "Tryptophan", 81, 151},
    new Object[] { "Seafood", "Cod", "Tryptophan", 90, 119},
    new Object[] { "Fruits", "Apples", "Manganese", 33, 61},
    new Object[] { "Fruits", "Cantaloupe", "Vitamin C", 112, 56},
    new Object[] { "Fruits", "Grapes", "Manganese", 33, 61},
    new Object[] { "Fruits", "Pineapple", "Manganese", 128, 75},
    new Object[] { "Fruits", "Strawberries", "Vitamin C", 24, 48},
    new Object[] { "Fruits", "Watermelon", "Vitamin C", 24, 48},
    new Object[] { "Poultry & Lean Meats", "Beef, lean organic",
    "Tryptophan", 112, 240},
    new Object[] { "Poultry & Lean Meats", "Lamb", "Tryptophan", 109,
229},
    new Object[] { "Poultry & Lean Meats", "Chicken", "Tryptophan",
121, 223},
    new Object[] { "Poultry & Lean Meats", "Venison ", "Protein", 69,
179},
    new Object[] { "Grains", "Corn ", "Vitamin B1", 24, 177},
    new Object[] { "Grains", "Oats ", "Manganese", 69, 147},
    new Object[] { "Grains", "Barley ", "Dietary Fiber", 54, 270}
};

}

```

Then, we can make it a groups model by extending from GroupsModelArray^[5]:

```

//GroupsModel
package foo;
public class FoodGroupsModel extends GroupsModelArray {
    public FoodGroupsModel(java.util.Comparator cmpr) {
        super(_foods, cmpr); //assume we
        //cmpr is used to group
    }
    protected Object createGroupHead(Object[] groupdata, int index, int
col) {
        return ((Object[])groupdata[0])[col];
        //groupdata is one of groups after GroupsModelArray groups
_foods
        //here we pick the first element and use the col-th column as
the group head
    }
    private static Object[][] _foods = new Object[][] {
        //...tabular data as shown above
    };
}

```

```
};
```

In addition, we have to implement a comparator to group the data based on the given column as follows.

```
package foo;
public class FoodComparator implements java.util.Comparator {
    int _col;
    boolean _asc;
    public FoodComparator(long col, boolean asc) {
        _col = (int) col; //which column to compare
        _asc = asc; //ascending or descending
    }
    public int compare(Object o1, Object o2) {
        Object[] data = (Object[]) o1;
        Object[] data2 = (Object[]) o2;
        int v = ((Comparable) data[_col]).compareTo(data2[_col]);
        return _asc ? v : -v;
    }
}
```

Since the data will be displayed in multiple columns, we have to implement a renderer. Here is an example.

```
public class FoodGroupRenderer implements RowRenderer {
    public void render(Row row, java.lang.Object obj, int index) {
        if (row instanceof Group) {
            //display the group head
            row.appendChild(new Label(obj.toString()));
        } else {
            //display an element
            Object[] data = (Object[]) obj;
            row.appendChild(new Label(data[0].toString()));
            row.appendChild(new Label(data[1].toString()));
            row.appendChild(new Label(data[2].toString()));
            row.appendChild(new Label(data[3].toString()));
            row.appendChild(new Label(data[4].toString()));
        }
    }
};
```

Finally we could group them together in a ZUML document as follows.

```
<?taglib uri="http://www.zkoss.org/dsp/web/core" prefix="c" ?>
<grid rowRenderer="${c:new('foo.FoodGroupRenderer')}"
      model="${c:newl('foo.FoodGroupsModel', c:new2('foo.FoodComparator',
0, true))}">
    <!-- Initially, we group data on 1st column in ascending order -->
    <columns menupopup="auto"> <!-- turn on column's menupopup -->
        <column label="Category"
               sortAscending="${c:new2('foo.FoodComparator', 0, true)}"
               sortDescending="${c:new2('foo.FoodComparator', 0, false)}" />
```

```

        sortDirection="ascending"/> <!-- since it is initialized as sorted -->
<column label="Name"
        sortAscending="${c:new2('foo.FoodComparator', 1, true)}"
        sortDescending="${c:new2('foo.FoodComparator', 1, false)}/>
<column label="Top Nutrients"
        sortAscending="${c:new2('foo.FoodComparator', 2, true)}"
        sortDescending="${c:new2('foo.FoodComparator', 2, false)}/>
<column label="% of Daily"
        sortAscending="${c:new2('foo.FoodComparator', 3, true)}"
        sortDescending="${c:new2('foo.FoodComparator', 3, false)}/>
<column label="Calories"
        sortAscending="${c:new2('foo.FoodComparator', 4, true)}"
        sortDescending="${c:new2('foo.FoodComparator', 4, false)}/>
</columns>
</grid>
```

If it is not the behavior you want, you could override `java.lang.Object[]`^[13], `java.util.Comparator`, `boolean`, `int` `GroupsModelArray.sortGroupData(java.lang.Object, java.lang.Object[], java.util.Comparator, boolean, int)`. Of course, you could extend from `AbstractGroupsModel`^[3] to have total control.

5.0.6 and Later

Since 5.0.6, it is much easier to handle tabular data:

First, `[[14], int, int] GroupsModelArray.createGroupHead(java.lang.Object[], int, int)` will return the correct element, so you don't have to override it as shown above.

Second, `ArrayComparator`^[15] was introduced, so `foo.FoodComparator` is not required in the above example.

Third, `Column.setSort(java.lang.String)`^[16] supports `auto(0)`, `auto(1)`, etc.

Thus, we can simplify the above example as follows.

```

<grid apply="foo.FoodComposer">
    <columns menupopup="auto"> <!-- turn on column's menupopup -->
        <column label="Category" sort="auto(0)"
            sortDirection="ascending"/> <!-- since it is initialized as sorted -->
        <column label="Name" sort="auto(1)"/>
        <column label="Top Nutrients" sort="auto(2)"/>
        <column label="% of Daily" sort="auto(3)"/>
        <column label="Calories" sort="auto(4)"/>
    </columns>
</grid>
```

And, the composer is as follows.

```

package foo;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.util.Composer;
import org.zkoss.zul.*;
public class FoodComposer implements Composer {
    public void doAfterCompose(Component comp) throws Exception {
        Grid grid = (Grid)comp;
```

```

        grid.setModel(new GroupsModelArray(_foods, new
ArrayComparator(0, true)));
        //Initially, we group data on 1st column in ascending
order
        grid.setRowRenderer(new FoodGroupRenderer());
    }
}
}

```

Example: Grouping Array of JavaBean

Suppose you have a collection of JavaBean objects (i.e., with the proper getter methods) as follows.

```

public class Food {
    String _category, _name, _nutrients;
    int _percentageOfDaily, _calories;

    public Food(String cat, String nm, String nutr, int pod, int cal) {
        _category = cat;
        _name = nm;
        _nutrients = nutr;
        _percentageOfDaily = pod;
        _calories = cal;
    }

    public String getCategory() {
        return _category;
    }

    public String getName() {
        return _name;
    }

    public String getNutrients() {
        return _nutrients;
    }

    public int getPercentageOfDaily() {
        return _percentageOfDaily;
    }

    public int getCalories() {
        return _calories;
    }
}

```

Assume you want to use the value of the field that the user uses to group the data, then you could override GroupsModelArray^[5] as follows.

```

public class FoodGroupsModel extends GroupsModelArray {
    public FoodGroupsModel(Food[] foods) {
        super(foods, new FieldComparator("category", true));
    }

    protected Object createGroupHead(Object[] groupdata, int index, int
col) {

```

```

        return new Object[] {groupdata[0], new Integer(col)};
    }
};

```

where

- We use FieldComparator^[17] to initialize the groups at the category field.
- We use an object array as the group head that carries the first element of the given group (Food[]), and the index of the column that causes the grouping. We will use the index later to retrieve the field's value

We also need a custom renderer:

```

package foo;
import org.zkoss.lang.reflect.Fields;
import org.zkoss.zk.ui.*;
import org.zkoss.zul.*;
public class FoodGroupRenderer implements RowRenderer {
    public void render(Row row, java.lang.Object obj, int index) {
        if (row instanceof Group) {
            Object[] data = (Object[])obj; //prepared by
createGroupHead()
            row.appendChild(new Label(getGroupHead(row, (Food)data[0],
(Integer)data[1])));
        } else {
            Food food = (Food) obj;
            row.appendChild(new Label(food.getCategory()));
            row.appendChild(new Label(food.getName()));
            row.appendChild(new Label(food.getNutrients()));
            row.appendChild(new Label(food.getPercentageOfDaily() +
""));
            row.appendChild(new Label(food.getCalories() + ""));
        }
    }
    private String getGroupHead(Row row, Food food, int index) {
        Column column =
(Column)row.getGrid().getColumns().getChildren().get(index);
        String orderBy =
((FieldComparator)column.getSortAscending()).getOrderBy();
        int j = orderBy.indexOf("name="),
            k = orderBy.indexOf(' ');
        try {
            return Fields.get(food, orderBy.substring(j+1, k>0 ? k:
orderBy.length())).toString();
        } catch (NoSuchMethodException ex) {
            throw UiException.Aide.wrap(ex);
        }
    }
};

```

The retrieval of the field's value is a bit tricky: since we will use `auto(fieldName)` to group and sort data for a given column (see the ZUML content listed below), we could retrieve the field's name by use of `FieldComparator.getOrderBy()`^[18], which returns something like "name=category ASC". Then, use `java.lang.String` `Fields.get(java.lang.Object, java.lang.String)`^[19] to retrieve it. If the field name is in a compound format, such as `something.yet.another`, you could use `java.lang.String` `Fields.getByCompound(java.lang.Object, java.lang.String)`^[20]

For 5.0.6 or later, you could use `FieldComparator.getRawOrderBy()`^[21] instead, which returns the field name you passed to `Column.setSort(java.lang.String)`^[16], i.e., "category".

```
Column column =
(Column) row.getGrid().getColumns().getChildren().get(index);
String field =
((FieldComparator)column.getSortAscending()).getRawOrderBy();
return Fields.get(food, field).toString();
```

Then, you could have the ZUML document as follows.

```
<grid apply="foo.FoodComposer">
<columns menupopup="auto">
<column label="Category" sort="auto(category)" sortDirection="ascending"/>
<column label="Name" sort="auto(name)"/>
<column label="Top Nutrients" sort="auto(nutrients)"/>
<column label="% of Daily" sort="auto(percentageOfDaily)"/>
<column label="Calories" sort="auto(calories)"/>
</columns>
</grid>
```

And, the composer is as follows.

```
package foo;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.util.Composer;
import org.zkoss.zul.*;
public class FoodComposer implements Composer {
    Food[] _foods = new Food[] { //assume we have a collection of foods
        new Food("Vegetables", "Asparagus", "Vitamin K", 115, 43),
        new Food("Vegetables", "Beets", "Folate", 33, 74)
        //...more
    };
    public void doAfterCompose(Component comp) throws Exception {
        Grid grid = (Grid)comp;
        grid.setModel(new FoodGroupsModel(_foods));
        //Initially, we group data on 1st column in ascending order
        grid.setRowRenderer(new FoodGroupRenderer());
    }
}
```

Group Foot

If the groups model supports a foot (such as a summary of all data in the same group), you could return an object to represent the footer when GroupsModel.getGroupfoot(int) [22] is called (similar to GroupsModel.getGroup(int) [23] shall return an object representing the group).

If you use GroupsModelArray [5], you could override [24], int, int) GroupsModelArray.createGroupFoot(java.lang.Object[], int, int)]. For example,

```
public class FoodGroupsModel extends GroupsModelArray {
    protected Object createGroupFoot(Object[] groupdata, int index, int col) {
        return "Total " + groupdata.length + " items";
    }
    ...
}
```

Version History

Version	Date	Content
5.0.6	December 2010	Enhanced the support of tabular data as described in #5.0.6 and Later.

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/GroupsModel.html#>
- [2] <http://www.zkoss.org/zkdemo/grid/grouping>
- [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/AbstractGroupsModel.html#>
- [4] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/SimpleGroupsModel.html#>
- [5] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/GroupsModelArray.html#>
- [6] <http://www.zkoss.org/javadoc/5.0.4/zk/org/zkoss/zul/ArrayGroupsModel.html>
- [7] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/GroupsSortableModel.html#>
- [8] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/GroupsSortableModel.html#group\(java.util.Comparator,boolean,int\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/GroupsSortableModel.html#group(java.util.Comparator,boolean,int))
- [9] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/GroupsSortableModel.html#sort\(java.util.Comparator,boolean,int\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/GroupsSortableModel.html#sort(java.util.Comparator,boolean,int))
- [10] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/GroupComparator.html#>
- [11] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Column.html#setSortAscending\(java.util.Comparator\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Column.html#setSortAscending(java.util.Comparator))
- [12] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Column.html#setSortDescending\(java.util.Comparator\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Column.html#setSortDescending(java.util.Comparator))
- [13] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/GroupsModelArray.html#sortGroupData\(java.lang.Object,java.util.Comparator\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/GroupsModelArray.html#sortGroupData(java.lang.Object,java.util.Comparator))
- [14] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/GroupsModelArray.html#createGroupHead\(java.lang.Object,java.util.Comparator\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/GroupsModelArray.html#createGroupHead(java.lang.Object,java.util.Comparator))
- [15] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ArrayComparator.html#>
- [16] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Column.html#setSort\(java.lang.String,java.util.Comparator\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Column.html#setSort(java.lang.String,java.util.Comparator))
- [17] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/FieldComparator.html#>
- [18] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/FieldComparator.html#getOrderBy\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/FieldComparator.html#getOrderBy())
- [19] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/lang/reflect/Fields.html#get\(java.lang.Object,java.util.Comparator\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/lang/reflect/Fields.html#get(java.lang.Object,java.util.Comparator))
- [20] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/lang/reflect/Fields.html#getByCompound\(java.lang.Object,java.util.Comparator\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/lang/reflect/Fields.html#getByCompound(java.lang.Object,java.util.Comparator))
- [21] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/FieldComparator.html#getRawOrderBy\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/FieldComparator.html#getRawOrderBy())
- [22] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/GroupsModel.html#getGroupfoot\(int\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/GroupsModel.html#getGroupfoot(int))
- [23] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/GroupsModel.html#getGroup\(int\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/GroupsModel.html#getGroup(int))
- [24] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/GroupsModelArray.html#createGroupFoot\(java.lang.Object,java.util.Comparator\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/GroupsModelArray.html#createGroupFoot(java.lang.Object,java.util.Comparator))

Tree Model

Here we describe how to implement a tree model (TreeModel^[4]). You shall understand the interaction among a component, a model, and a renderer, please refer to the Model-driven Display section.

Choose a Proper Model Class

A TreeModel^[4] is the data model of a tree-like component, such as Tree^[1].

If the tree data is small enough to be loaded completely into a TreeModel, you can use DefaultTreeModel^[2] which accepts DefaultTreeNode^[3] to construct a tree^[4].

In a more complicated case, if you want to implement custom logic like load-on-demand and caching. Maybe the data is too large to load them all into a TreeModel at once. Then, we suggest you to extend AbstractTreeModel^[5], which will handle the data listeners transparently.

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Tree.html#>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeModel.html#>
- [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeNode.html#>
- [4] TreeModel (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/TreeModel.html#>) is available since 5.0.6. For 5.0.5 or prior, please use , which is similar except it assumes the tree structure is immutable
- [5] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/AbstractTreeModel.html#>

Example: In-Memory Tree with DefaultTreeModel

If you prefer to use TreeNode (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/TreeNode.html#>) to construct the tree dynamically, you can use DefaultTreeModel (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeModel.html#>) and DefaultTreeNode (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeNode.html#>). The usage is straightforward, but it means that the whole tree must be constructed before it is displayed.

For example, suppose we want to show a tree of file information, and the file information is stored as FileInfo:

```
public class FileInfo {
    public final String path;
    public final String description;
    public FileInfo(String path, String description) {
        this.path = path;
        this.description = description;
    }
}
```

Then, we can create a tree of file information with DefaultTreeModel (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeModel.html#>) as follows:

```
TreeModel model = new DefaultTreeModel(
    new DefaultTreeNode(null,
        new DefaultTreeNode[] {
            new DefaultTreeNode(new FileInfo("/doc", "Release and License
Notes")),
            new DefaultTreeNode(new FileInfo("/dist", "Distribution")),
    })
);
```

```

        new DefaultTreeNode[] {
            new DefaultTreeNode(new FileInfo("/lib", "ZK Libraries"),
                new DefaultTreeNode[] {
                    new DefaultTreeNode(new FileInfo("zcommon.jar", "ZK
Common Library")),
                    new DefaultTreeNode(new FileInfo("zk.jar", "ZK Core
Library"))
                },
                new DefaultTreeNode(new FileInfo("/src", "Source Code")),
                new DefaultTreeNode(new FileInfo("/xsd", "XSD Files"))
            )
        }
    );
}

```

Here, we render `FileInfo` in a custom renderer:

```

import org.zkoss.zul.*;
public class FileInfoRenderer implements TreeitemRenderer<DefaultTreeNode<FileInfo>> {
    public void render(Treeitem item, DefaultTreeNode<FileInfo> data, int index)
throws Exception {
    FileInfo fi = data.getData();
    Treerow tr = new Treerow();
    item.appendChild(tr);
    tr.appendChild(new Treecell(fi.path));
    tr.appendChild(new Treecell(fi.description));
}
}

```

Next, bind them together in a composer:

```

public class FileInfoTreeController extends SelectorComposer {
    @Wire
    private Tree tree;

    @Override
    public void doAfterCompose(Div div) throws Exception{
        super.doAfterCompose(div);
        tree.setModel(new DefaultTreeModel(.../*as shown above*/));
        tree.setItemRenderer(new FileInfoRenderer());
    }
}

```

Then, we can put them together in a ZUML document:

```

<div apply="org.zkoss.reference.developer.mvc.model.FileInfoTreeController">
    <tree id="tree">
        <treetcols>
            <treetcol label="Path"/>
            <treetcol label="Description"/>
        </treetcols>

```

```
</tree>
</div>
```

Then, the result:

Path	Description
/doc	Release and License Notes
▲ /dist	Distribution
▲ /lib	ZK Libraries
zcommon.jar	ZK Common Library
zk.jar	ZK Core Library
/src	Source Code
/xsd	XSD Files

Notice that you can manipulate the tree dynamically (such as adding a node with DefaultTreeNode.add(org.zkoss.zul.TreeNode) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeNode.html#add\(org.zkoss.zul.TreeNode\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeNode.html#add(org.zkoss.zul.TreeNode)))). The tree shown at the browser will be modified accordingly.

Example: Create/Update/Delete operation with DefaultTreeNode

The benefit of using DefaultTreeNode is it will notify Tree (or the component associated with a TreeModel) about the node change including adding, deleting, and updating. (The underlying implementation is DefaultTreeNode will fire an event when you call its add(), insert(), setData(), removeFromParent()).

To demonstrate the example, first we add create, update and delete buttons in a .zul:

```
<tree id="tree">
...
</tree>
<grid>
  <auxhead>
    <auxheader colspan="2" label="Add/Edit FileInfo" />
  </auxhead>
  <columns visible="false">
    <column />
    <column />
  </columns>
  <rows>
    <row>
      <cell><textbox id="pathTbx" /></cell>
      <cell><textbox id="descriptionTbx" width="300px"/></cell>
    </row>
    <row>
      <cell colspan="2" align="center">
        index: <intbox id="index" />
        <button id="create" label="Add to selected parent node" />
        <button id="update" label="update" />
      </cell>
    </row>
  </rows>
</grid>
```

```

        <button id="delete" label="delete" />
    </cell>
</row>
</rows>
</grid>
```

The intbox here is for specifying index to insert before the selected tree item.

Add/Insert

DefaultTreeNode (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeNode.html#>) provides DefaultTreeNode.add(org.zkoss.zulTreeNode) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeNode.html#add\(org.zkoss.zulTreeNode\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeNode.html#add(org.zkoss.zulTreeNode))) and int) DefaultTreeNode.insert(org.zkoss.zulTreeNode, int) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeNode.html#insert\(org.zkoss.zulTreeNode,int\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeNode.html#insert(org.zkoss.zulTreeNode,int))) that can manipulate the tree dynamically.

Here we register onClick event to create Button in `foo.FileInfoTreeController`:

```

//wire components as member fields
private Textbox pathTbx;
private Textbox descriptionTbx;
private Intbox index;
//register onClick event for creating new object into tree model
public void onClick$create() {
    String path = pathTbx.getValue();
    String description = descriptionTbx.getValue();
    if ("".equals(path)) {
        alert("no new content to add");
    } else {
        Treeitem selectedTreeItem = tree.getSelectedItem();
        DefaultTreeNode newNode = new DefaultTreeNode(new
FileInfo(path, description));
        DefaultTreeNode selectedTreeNode = null;
        Integer i = index.getValue();
        // if no treeitem is selected, append child to root
        if (selectedTreeItem == null) {
            selectedTreeNode = (DefaultTreeNode) ((DefaultTreeModel)
tree.getModel()).getRoot();
            if (i == null) // if no index specified, append to last.
                selectedTreeNode.add(newNode);
            else // if index specified, insert before the index number.
                selectedTreeNode.insert(newNode, i);
        } else {
            selectedTreeNode = (DefaultTreeNode)
selectedTreeItem.getValue();
            if (selectedTreeNode.isLeaf())
                selectedTreeNode = selectedTreeNode.getParent();
            if (i == null)
                i = selectedTreeNode.getChildren().size();
        }
        selectedTreeNode.insert(newNode, i);
    }
}
```

```
        selectedTreeNode.add(newNode);  
    }  
    else  
        selectedTreeNode.insert(newNode, i);  
    }  
}  
}
```

If index is not specified, we add a new node using `DefaultTreeNode.add(org.zkoss.zul.TreeNode)` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeNode.html#add\(org.zkoss.zul.TreeNode\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeNode.html#add(org.zkoss.zul.TreeNode))) at the bottom of the parent node by default, or we can also use int `DefaultTreeNode.insert(org.zkoss.zul.TreeNode, int)` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeNode.html#insert\(org.zkoss.zul.TreeNode,%d\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeNode.html#insert(org.zkoss.zul.TreeNode,%d))) to insert a new node before the specified index.

Update/Delete

DefaultTreeNode (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeNode.html#>) provides DefaultTreeNode.setData(java.lang.Object) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeNode.html#setData\(java.lang.Object\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeNode.html#setData(java.lang.Object))) which can update selected tree items and DefaultTreeNode.removeFromParent() ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeNode.html#removeFromParent\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeNode.html#removeFromParent())) that can delete the selected tree item from its parent node.

Here we register onClick event to update and delete Button in `foo.FileInfoTreeController`:

```
//register onClick event for updating edited data in tree model
public void onClick$update() {
    Treeitem selectedTreeItem = treeGrid.getSelectedItem();
    if(selectedTreeItem == null) {
        alert("select one item to update");
    } else {
        DefaultTreeNode selectedTreeNode = (DefaultTreeNode)
selectedTreeItem.getValue();
        //get current FileInfo from selected tree node
        FileInfo fileInfo = (FileInfo) selectedTreeNode.getData();
        //set new value of current FileInfo
        fileInfo.setPath(pathTbx.getValue());
        fileInfo.setDescription(descriptionTbx.getValue());
        //set current FileInfo in the selected tree node
        selectedTreeNode.setData(fileInfo);
    }
}

//register onClick event for removing data in tree model
public void onClick$delete() {
    final Treeitem selectedTreeItem = treeGrid.getSelectedItem();
    if(selectedTreeItem == null) {
        alert("select one item to delete");
    } else {
        DefaultTreeNode selectedTreeNode = (DefaultTreeNode)
selectedTreeItem.getValue();
        selectedTreeNode.removeFromParent();
    }
}
```

```

    }
}
```

For updating tree node data, we have to modify render() of foo.FileInfoRenderer:

```

public void render(Treeitem item, DefaultTreeNode<FileInfo> data, int index)
throws Exception {
    FileInfo fi = data.getData();
    if (tr == null) {
        tr = new Treerow();
    } else {
        tr.getChildren().clear();
    }
    item.appendChild(tr);
    tr.appendChild(new Treecell(fi.getPath()));
    tr.appendChild(new Treecell(fi.getDescription()));
}
```

Example: Load-on-Demand Tree with AbstractTreeModel

Implementing all TreeModel (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/TreeModel.html#>) directly provides the maximal flexibility, such as load-on-demand and caching. For example, you don't have to load a node until int) TreeModel.getChild(java.lang.Object, int) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/TreeModel.html#getChild\(java.lang.Object,%20int\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/TreeModel.html#getChild(java.lang.Object,%20int))) is called. In addition, you could load and cache all children of a given node when int) TreeModel.getChild(java.lang.Object, int) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/TreeModel.html#getChild\(java.lang.Object,%20int\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/TreeModel.html#getChild(java.lang.Object,%20int))) is called the first time against a particular node, and then return a child directly if it is in the cache.

For example (pseudo code):

```

public class MyModel extends AbstractTreeModel<Object> {
    public Object getChild(Object parent, int index) {
        Object[] children = _cache.get(parent); //assume you have a
        cache for children of a given node
        if (children == null)
            children = _cache.loadChildren(parent); //ask cache to load
        all children of a given node
        return children[index];
    }
    ...
}
```

By extending from AbstractTreeModel (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/AbstractTreeModel.html#>), you have to implement three methods: int) TreeModel.getChild(java.lang.Object, int) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/TreeModel.html#getChild\(java.lang.Object,%20int\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/TreeModel.html#getChild(java.lang.Object,%20int))), TreeModel.getChildCount(java.lang.Object) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/TreeModel.html#getChildCount\(java.lang.Object\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/TreeModel.html#getChildCount(java.lang.Object))), and TreeModel.isLeaf(java.lang.Object) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/TreeModel.html#isLeaf\(java.lang.Object\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/TreeModel.html#isLeaf(java.lang.Object))). Optionally, you could implement java.langObject) TreeModel.getIndexOfChild(java.lang.Object, java.langObject) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/TreeModel.html#getIndexOfChild\(java.lang.Object,%20java.langObject\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/TreeModel.html#getIndexOfChild(java.lang.Object,%20java.langObject))^[1], if you have a better algorithm than iterating through all children of a given parent.

Improving Performance

You **should** override `TreeModel.getPath(E)` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/TreeModel.html#getPath\(E\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/TreeModel.html#getPath(E))) to implement an efficient way to deduce the sibling index of each ancestor of a node. Because the default implementation of `getPath()` in `AbstractTreeModel` will traverse from the root node to compute the index which is inefficient with lots of nodes. Such node traversing will call `getChild()` and load unnecessary nodes from the data source which consume more memory.

An Example

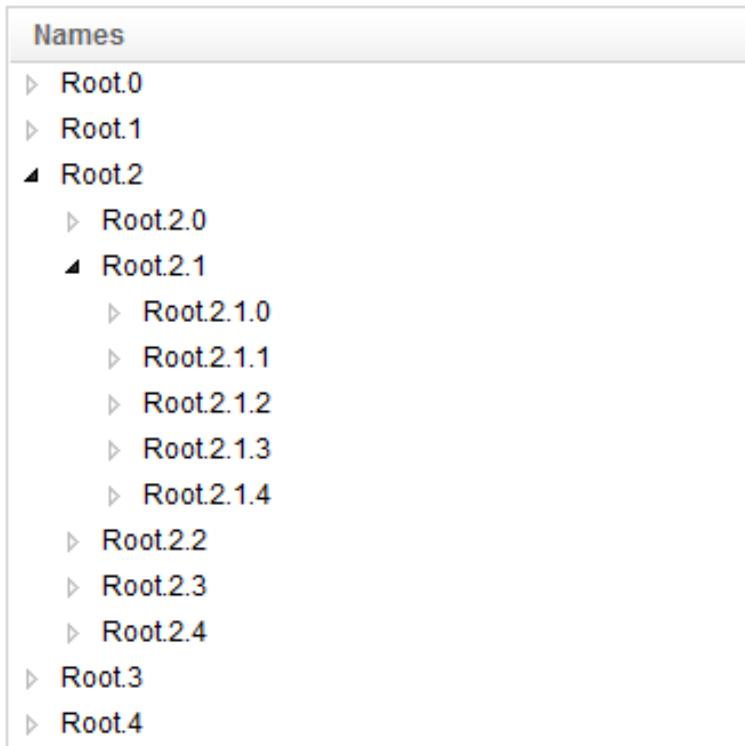
Here is a simple example, which generates a 4-level tree and each branch has 5 children:

```
package foo;
public class LoadOnDemandModel extends AbstractTreeModel<Object> {
    public LoadOnDemandModel() {
        super("Root");
    }
    public boolean isLeaf(Object node) {
        return getLevel((String)node) >= 4; //at most 4 levels
    }
    public Object getChild(Object parent, int index) {
        return parent + "." + index;
    }
    public int getChildCount(Object parent) {
        return isLeaf(parent) ? 0 : 5; //each node has 5 children
    }
    public int getIndexOfChild(Object parent, Object child) {
        String data = (String)child;
        int i = data.lastIndexOf('.');
        return Integer.parseInt(data.substring(i + 1));
    }
    private int getLevel(String data) {
        for (int i = -1, level = 0;; ++level)
            if ((i = data.indexOf('.', i + 1)) < 0)
                return level;
    }
}
```

Then, we assign this model to a tree:

```
<?taglib uri="http://www.zkoss.org/dsp/web/core" prefix="c" ?>
<tree model="${c:new('org.zkoss.reference.developer.mvc.model.LoadOnDemandModel')}>
    <treetcols>
        <treetcol label="Names"/>
    </treetcols>
</tree>
```

And, the result looks like this:



Sorting

Interface: Sortable (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/Sortable.html#>)

Implementation: You have to implement it explicitly

To support the sorting, the model must implement Sortable (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/Sortable.html#>) too. Thus, when the end user clicks the header to request the sorting, boolean) Sortable.sort(java.util.Comparator, boolean) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/Sortable.html#sort\(java.util.Comparator,boolean\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/Sortable.html#sort(java.util.Comparator,boolean))) will be called.

For example, (pseudo code)

```
public class FooModel extends AbstractTreeModel implements Sortable {  
    public void sort(Comparator cmpr, final boolean ascending) {  
        sortData(cmpr); //sort your data here  
        fireEvent(ListDataEvent.CONTENT_CHANGED, -1, -1); //ask  
component to reload all  
    }  
    ...  
}
```

Notice that the ascending parameter is used only for reference and you usually don't need it, since the cmpr is already a comparator capable to sort in the order specified in the ascending parameter.

Selection

```
Interface: TreeSelectableModel (http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/TreeSelectableModel.html#)
Implementation: Implemented by AbstractTreeModel (http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/AbstractTreeModel.html#)
```

If your data model also provides the collection of selected elements, you shall also implement TreeSelectableModel (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/TreeSelectableModel.html#>). When using with a component supporting the selection (such as Tree (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Tree.html#>)), the component will invoke [[http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/TreeSelectableModel.html#isPathSelected\(int\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/TreeSelectableModel.html#isPathSelected(int))] TreeSelectableModel.isPathSelected(int[]) to display the selected elements correctly. In addition, if the end user selects or deselects an item, [[http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/TreeSelectableModel.html#addSelectionPath\(int\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/TreeSelectableModel.html#addSelectionPath(int))] TreeSelectableModel.addSelectionPath(int[]) and [[http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/TreeSelectableModel.html#removeSelectionPath\(int\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/TreeSelectableModel.html#removeSelectionPath(int))] TreeSelectableModel.removeSelectionPath(int[]) will be called by the component to notify the model that the selection is changed. Then, you can update the selection into the persistent layer (such as database) if necessary.

On the other hand, when the model detects the selection is changed (such as [[http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/TreeSelectableModel.html#addSelectionPath\(int\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/TreeSelectableModel.html#addSelectionPath(int))] TreeSelectableModel.addSelectionPath(int[]]) is called), it has to fire the event, such as TreeDataEvent.SELECTION_CHANGED (http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/event/TreeDataEvent.html#SELECTION_CHANGED) to notify the component. It will cause the component to correct the selection^[2].

All default implementations, including AbstractTreeModel (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/AbstractTreeModel.html#>) and DefaultTreeModel (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeModel.html#>) implement TreeSelectableModel (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/TreeSelectableModel.html#>). Thus, your implementation generally doesn't have to implement it explicitly.

It is important to note that, once a tree is assigned with a tree model, the application shall not manipulate the tree items and/or change the selection of the tree directly. Rather, the application shall access only the list model to add, remove and select data elements. Let the model notify the component what has been changed.

[1] Tree (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Tree.html#>) is available in 5.0.6 and later.

[2] Don't worry. The component is smart enough to prevent the dead loop, even though the component invokes addSelectionPath() to notify the model while the model fires the event to notify the component.

Selection Control

With the multiple selection function in a data model, you have to implement a class for the SelectionControl (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/SelectionControl.html#>) to tell the data model which items are selectable and what it will perform a "select all" function with. The following implementation which extends DefaultSelectionControl (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/AbstractTreeModel/DefaultSelectionControl.html#>) is a simple example to change "selectable" items.

Please note that if your data model is much larger, you may implement on it your own to get rid of the performance impact.

```
model.setSelectionControl(new
AbstractTreeModel.DefaultSelectionControl(model) {
    public boolean isSelectable(Object e) {
        int i = model.indexOf(e);
```

```
    return i % 2 == 0;  
}  
});
```

Open Tree Nodes

Interface: TreeOpenableModel (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/TreeOpenableModel.html#>)
Implementation: Implemented by AbstractTreeModel (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/AbstractTreeModel.html#>)

By default, all tree nodes are closed. To control whether to open a tree node, you could implement TreeOpenableModel (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/TreeOpenableModel.html#>).

More importantly, to open a tree node, the application shall access the model's TreeOpenableModel (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/TreeOpenableModel.html#>) API, rather than accessing Treeitem (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Treeitem.html#>) directly.

All default implementations, including AbstractTreeModel (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/AbstractTreeModel.html#>) and DefaultTreeModel (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeModel.html#>) implement TreeOpenableModel (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/TreeOpenableModel.html#>). Thus, your implementation generally doesn't have to implement it explicitly.

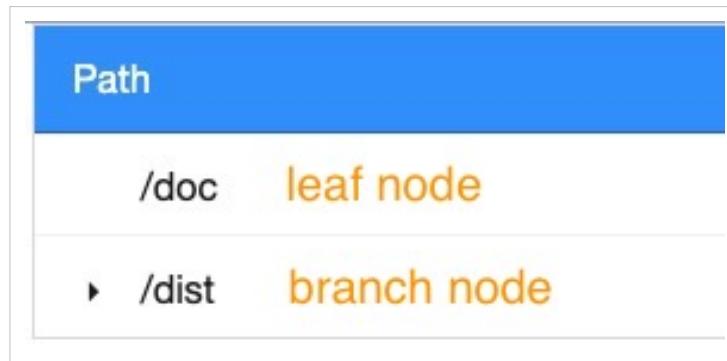
Note: If your tree model contains a lot of nodes, please also implement TreeModel.getPath(E) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/TreeModel.html#getPath\(E\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/TreeModel.html#getPath(E))) to get better performance, by default it is implemented by Depth-first search (http://en.wikipedia.org/wiki/Depth-first_search) to get the path from a tree node.

Leaf Node

The DefaultTreeNode (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeNode.html#>) has 2 constructors:

- DefaultTreeNode (data) : create a leaf node ([https://en.wikipedia.org/wiki/Tree_\(data_structure\)#Terminology](https://en.wikipedia.org/wiki/Tree_(data_structure)#Terminology)) which cannot have children added to it.
- DefaultTreeNode (data, children) : create a branch node ([https://en.wikipedia.org/wiki/Tree_\(data_structure\)#Terminology](https://en.wikipedia.org/wiki/Tree_(data_structure)#Terminology)) which can have children added to it.

ZK renders a rotating triangle to expand/collapse a node in front of a branch node, but a leaf node doesn't have that triangle.



If you want to display a leaf node, you should use DefaultTreeNode (data), otherwise even if you provide a zero-size list for DefaultTreeNode (data, children) constructor, ZK tree will still render the node as a branch node that contains no children. So when you expand the node, it shows nothing. It might confuse users.

DefaultTreeModel (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeModel.html#>)'s constructor accepts the 2nd boolean argument to determine whether to render a branch node without children as a leaf node.

```
DefaultTreeModel model2 = new DefaultTreeModel(root, true);
```

Version History

Version	Date	Content
5.0.6	January 2011	TreeNode (http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/TreeNode.html#), DefaultTreeNode (http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeNode.html#) and DefaultTreeModel (http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeModel.html#) were introduced.
6.0.0	February 2012	TreeSelectableModel (http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/TreeSelectableModel.html#) and TreeOpenableModel (http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/TreeOpenableModel.html#) were introduced to replace Selectable (Selectable.html#">http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext>Selectable.html#) and Openable (http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/Openable.html#).
5.0.12 / 6.0.3 / 6.5.1	October 2012	DefaultTreeModel (http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeModel.html#) adds a new constructor for configuring whether to treat the zero size of children node as a leaf node.

Chart Model

Here we describe how to implement a chart model (ChartModel^[1]). For the concept of component, model and render, please refer to the Model-driven Display section.

Depending on the type of chart you want, you could implement one of PieModel^[1], XYModel^[2], GanttModel^[3], HiLoModel^[4], etc. In addition, there are default implementations for them you could use directly, such as SimplePieModel^[4], SimpleXYModel^[5], etc.

For example, we could have a composer as follows.

```
public class ProgrammerModelComposer extends SelectorComposer<Component> {
    public void doAfterCompose(Component comp) throws Exception {
        PieModel piemodel = new SimplePieModel();
        piemodel.setValue("C/C++", new Double(12.5));
        piemodel.setValue("Java", new Double(50.2));
        piemodel.setValue("VB", new Double(20.5));
        piemodel.setValue("PHP", new Double(15.5));
        ((Chart) comp).setModel(piemodel);
    }
}
```

Then, you could use it in a ZUML document:

```
<chart title="Pie Chart" width="500" height="250" type="pie" threeD="false" fgAlpha="128"
apply="foo.ProgrammerModelComposer"/>
```

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/PieModel.html#>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/XYModel.html#>
- [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/GanttModel.html#>
- [4] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/HiLoModel.html#>
- [5] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/SimpleXYModel.html#>

Matrix Model

- Available for ZK:
- CE PE EE

Here we describe how to implement a matrix model (*MatrixModel*^[1]). For the concept of component, model and renderer, please refer to the Model-driven Display section.

By default, ZK does not provide a built-in model implementation class for *MatrixModel* because *Biglistbox* is designed to handle unlimited data set, therefore, there is no need to handle model data in memory. This usage is application-dependent and varies from case to case. However, you can extend your own implementation from the *AbstractListModel*^[6] skeleton class.

To implement a *MatrixModel*, one needs to consider the performance issue that handles a huge data set in memory with Java Collection Framework. The issue is when using the default implementation of *Java Collection Framework* as it goes through every entry to gather the value of **hashCode** when searching the key in **Map/Set** or to check every entry for **equals** and **toString** functions. This implementation method greatly reduces the performance of *Biglistbox*. Therefore, to use the *Biglistbox* component with *MatrixModel*, we need to implement a clever and simple *List* for traversing huge data sets.

FakerKeyList

In this example, we create a *FakerKeyList* to implement the *List* interface for *MatrixModel* to handle the partial big data in memory.

```
private class FakerKeyList<T> extends AbstractList<T> {
    final int _size;
    Map<String, T> _updateCache = new HashMap<String, T> ();
    final Fun<?> _fn;
    final String _key;

    public FakerKeyList(int size, int key, Fun<?> fn) {
        _size = size;
        _key = key + "_" + size;
        _fn = fn;
    }

    @Override
    public T get(int index) {
        // if changed, returns the changed value
        Object val = _updateCache.get(String.valueOf(index));
        if (val != null)
            return (T) val;
    }
}
```

```

        return (T) _fn.apply(index);
    }

    @Override
    public int hashCode() {
        return _key.hashCode();
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == this)
            return true;
        if (obj instanceof FakerKeyList) {
            return _key.equals(((FakerKeyList) (obj))._key);
        }
        return false;
    }

    @Override
    public String toString() {
        return _key;
    }

    // omitted...
}

```

As you can see, we use a **key** string as the key for *toString*, *hashCode*, and *equals* methods to speed up searching time. **Fun** class on the other hand, is a handy class to render the model data for this example.

FakerMatrixModel

In this example, we create a **FakerMatrixModel** to implement **MatrixModel**. Following is the fragment code:

```

public class FakerMatrixModel<Head extends List, Row extends List, Cell, Header> extends
AbstractListModel<Row> implements MatrixModel<Row, Head, Cell, Header>, Sortable {

    // omitted...

    private boolean _sortDir = true;

    @Override
    public Row getElementAt(int index) {
        final int rowIndex = _sortDir ? index : getSize() - index -
1; // handle the sorting
        final String key = String.valueOf(rowIndex);
        List<String> value = _rowCache.get(key);
        if (value == null) {
            value = new FakerKeyList<String>(_colSize, rowIndex, new
Fun() {

```

```
    @Override
    public Object apply(int index) {
        return "y = " + rowIndex;
    });
    _rowCache.put(key, value);
}
return (Row) value;
}

// omitted...
}
```

MatrixModel is extended from the **ListModel** interface and uses the **getElementAt(int)** method to receive row data from the **FakerKeyList** object that implements the **List** interface.

Sortable Model

The **MatrixModel** can also support Sortable [2] interface. In your implementor class you can just implement the Sortable [2] interface and provide boolean) Sortable.sort(java.util.Comparator, boolean) [3] and Sortable.getSortDirection(java.util.Comparator) [4] methods.

For example,

```
public void sort(Comparator cmpr, boolean ascending) {
    _sorting = cmpr;
    _sortDir = ascending;
    fireEvent(ListDataEvent.STRUCTURE_CHANGED, -1, -1);
}

@Override
public String getSortDirection(Comparator cmpr) {
    if (Objects.equals(_sorting, cmpr))
        return _sortDir ? "ascending" : "descending";
    return "natural";
}
```

As you can see, we fire a data change event with *ListDataEvent.STRUCTURE_CHANGED* attribute to notify the component that model data has been changed.

Notify for Data Updates

MatrixModel is the same as *ListModel* when notify for data updates, please refer to `ListMode#Notify_for_Data_Updates`

Resource

All of the examples above can be found here - Github's FakerMatrixModel source code ^[5]

Version History

Version	Date	Content
6.0.1	March 2012	The Biglistbox and MatrixModel were introduced

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkmax/zul/MatrixModel.html#>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/Sortable.html#>
- [3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/Sortable.html#sort\(java.util.Comparator,](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/Sortable.html#sort(java.util.Comparator,)
- [4] [>getSortDirection\(java.util.Comparator\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/Sortable.html#getSortDirection(java.util.Comparator))
- [5] <https://github.com/zkoss/zk/blob/master/zktest/src/org/zkoss/zktest/test2/big/FakerMatrixModel.java>

View

The view is the UI of an application. It totally depends on the application's requirements.

As described in MVC/Model, some ZK components support Model-driven rendering, such as Listbox ^[3]. There are two approaches you can customize the rendering of each item in a model:

- Template: you define a template which is a fragment of the ZUML document to define how to render each item. It's more readable and easy to use.
 - Renderer: you create a Java class that implements a specific interface to render each item. If you need to render items according to conditions in the runtime, this approach is suggested.
-

Template

A template is a ZUML fragment that defines how to create components. A template is enclosed with the `template` element as shown below.

```
<window>
    <template name="foo">
        <textbox/>
        <grid model=${data}>
            <columns/>
            <template name="model"> <!-- nested template -->
                <row>Name: <textbox value="${each.name}" /></row>
            </template>
        </grid>
    </template>
    ...

```

A template can contain any ZUML elements you want, including other templates. When a ZUML document is interpreted, a template won't be interpreted immediately. Rather, it will be encapsulated as an instance of `Template`^[1], and be associated to a component. Then, the component or a tool can create the components repeatedly based on the template by invoking `org.zkoss.zk.ui.Component`, `org.zkoss.xel.VariableResolver`, `org.zkoss.zk.ui.util.Composer` `Template.create(org.zkoss.zk.ui.Component, org.zkoss.zk.ui.Component, org.zkoss.xel.VariableResolver, org.zkoss.zk.ui.util.Composer)`^[2].

A component can be assigned with multiple templates. Each of them is identified by the `name` attribute.

```
<div>
    <template name="t1">
        <grid model="${foo}" />
    </template>
    <template name="t2">
        <listbox model="${foo}" />
    </template>

```

How a template is used depends on the component it associates with and the tools you use. Currently, all components that support the concept of `model` allow you to specify a template to control how to render each item. In the following sections, we discuss them in details. If you'd like to know how to use templates manually in Java, please refer to the UI Patterns: Templates section.

Notice that please read the Listbox Template section first, even though you're rendering other kind of UI. It described the common concepts and tricks of using templates.

References

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

[2] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Template.html#create\(org.zkoss.zk.ui.Component, org.zkoss.xel.VariableResolver, org.zkoss.zk.ui.util.Composer\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Template.html#create(org.zkoss.zk.ui.Component, org.zkoss.xel.VariableResolver, org.zkoss.zk.ui.util.Composer))

Listbox Template

The template used to control the rendering of each item must be named `model` and declared right inside the `listbox` element. For example,

```
<div apply="foo.FruitProvider">
    <listbox model="${$composer.fruits}">
        <listhead>
            <listheader label="Name" sort="auto"/>
            <listheader label="Weight" sort="auto"/>
        </listhead>
        <template name="model">
            <listitem>
                <listcell label="${each[0]}"/>
                <listcell label="${each[1]}"/>
            </listitem>
        </template>
    </listbox>
</div>
```

The template's name is important because users are allowed to associate multiple templates to one component, and `listbox`'s default renderer looks only for the template called `model`.

When the template is rendered, a variable called `each` is assigned with the data being rendered. Thus, you could retrieve the information to render with EL expressions, such as `${each[0]}`, if it is an array, or `${each.name}`, if it is a bean with a getter called `name`.

In this example, we assume the `$composer.fruits` expression returns a two-dimensional array^[1], and is provided by the `foo.FruitProvider` composer such as follows^[2].

```
public class FruitProvider extends
org.zkoss.zk.ui.select.SelectorComposer {
    public ListModelArray fruits = new ListModelArray(
        new String[][] {
            {"Apple", "10kg"},
            {"Orange", "20kg"},
            {"Mango", "12kg"}
        });
    public ListModelArray getFruits() {
        return fruits;
    }
}
```

Apple	10kg
Orange	20kg
Mango	12kg

-
- [1] Of course, it can be anything you like. Just make sure it matches the EL expressions specified in the template.
 - [2] Here we use UNIQ-javadoc-0-736dd3f9533a2ca2-QINU for simplicity. There are several ways to implement a composer, such as wiring a Spring-managed bean. For more information, please refer to the Composer section.

Component's Value

By default, the data used to render a component will be stored to the component's value property automatically. For listitem, it is Listitem.setValue(T) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Listitem.html#setValue\(T\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Listitem.html#setValue(T))). Thus, you retrieve it back easily by invoking Listitem.getValue() ([Of course, if you prefer to store other values, you can simply specify `value="\${whatever}"` to the listitem element in the template.](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul>Listitem.html#getValue()).</p></div><div data-bbox=)

The forEachStatus Variable

There is a variable called forEachStatus providing the information of the iteration. It is an instance of ForEachStatus (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ForEachStatus.html#>). For example, you could retrieve the iteration's index by use of `${forEachStatus.index}`.

Lifecycle and the arg Variable

When using the template, it is important to remember that the template is rendered on demand. It means the template can be rendered very late, after the page is rendered, after the user scrolls down to make an item visible, and so on. Thus, in the template, you *cannot* reference anything that is available only in the page rendering phase. For example, you can't reference the arg variable in a template:

```
<listbox model="${$composer.fruits}" apply="foo.FruitProvider">
    <template name="model">
        <listitem>
            <listcell label="${arg.foo}" /> <!-- Wrong! it is always empty -->
            <listcell label="${each}" />
        </listitem>
    </template>
</listbox>
```

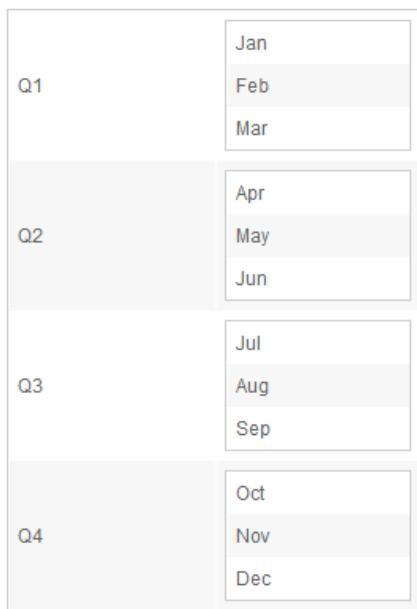
To work around, you have to store the value in, say, component's custom attributes (Component.getAttributes() ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Component.html#getAttributes\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Component.html#getAttributes()))). For example,

```
<listbox model="${$composer.fruits}" apply="foo.FruitProvider">
    <custom-attributes foo="${arg.foo}" /><!-- store it for later use -->
    <template name="model">
        <listitem>
            <listcell label="${foo}" /> <!-- Correct! Use the stored copy. -->
            <listcell label="${each}" />
        </listitem>
    </template>
</listbox>
```

Nested Listboxes

The template can be applied recursively. Here is an example of a listbox-in-listbox:

```
<zk>
<zscript><! [CDATA[
ListModel quarters = newListModelArray(new String[] {"Q1", "Q2",
"Q3", "Q4"});
Map months = new HashMap();
months.put("Q1", newListModelArray(new String[] {"Jan", "Feb",
"Mar"}));
months.put("Q2", newListModelArray(new String[] {"Apr", "May",
"Jun"}));
months.put("Q3", newListModelArray(new String[] {"Jul", "Aug",
"Sep"}));
months.put("Q4", newListModelArray(new String[] {"Oct", "Nov",
"Dec"}));
ListModel qs = (quarters);
]]></zscript>
<listbox model="${quarters}">
<template name="model">
<listitem>
<listcell>${each}</listcell>
<listcell>
<listbox model="${months[each]}">
<template name="model">
<listitem label="${each}" />
</template>
</listbox>
</listcell>
</listitem>
</template>
</listbox>
</zk>
```



How to retrieve the outer template's data in the inner template

Although `forEachStatus` has an API called `ForEachStatus.getPrevious()` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ForEachStatus.html#getPrevious\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ForEachStatus.html#getPrevious())), it always returns null^[1]. It is because the template is rendered on demand. When ZK is rendering the inner template, the previous iteration has already gone. There is no way to retrieve the iteration information of the outer template.

Rather, you have to traverse the component tree or use the `custom-attributes` element.

Here is an example of traversing the component tree to retrieve the data in the outer template, as shown at line 9 below. Notice that, each data is, as described before, stored in the component's `value` property.

```
<listbox model="${quarters}">
    <template name="model">
        <listitem>
            <listcell>
                <listbox model="${months[each]}">
                    <template name="model">
                        <listitem>
                            <listcell label="${forEachStatus.index}" />

                            <listcell>${self.parent.parent.parent.parent.value}</listcell>
                            <listcell>${each}</listcell>
                        </listitem>
                    </template>
                </listbox>
            </listcell>
        </listitem>
    </template>
</listbox>
```

If the component tree is deep, It is tedious and somehow error prone. Alternatively, you can store the information into a custom attribute and then retrieve it later, as shown at line 4 and 10 below.

```

<listbox model="${quarters}">
    <template name="model">
        <listitem>
            <custom-attributes master="${each}" />
            <listcell>
                <listbox model="${months[each]}>
                    <template name="model">
                        <listitem>
                            <listcell label="${forEachStatus.index}" />
                            <listcell>${master}</listcell>
                            <listcell>${each}</listcell>
                        </listitem>
                    </template>
                </listbox>
            </listcell>
        </listitem>
    </template>
</listbox>

```

[1] On the other hand, it returns the previous iteration information when using with the forEach attribute.

Template for GroupsModel

When used with GroupsModel (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/GroupsModel.html#>), listboxes will use the template called `model:group` for rendering the grouping object. If it is not defined, it will look for the template called `model` instead (i.e., the same template is used for rendering the grouping and non-grouping objects).

```

<listbox model="${fooGroupsModel}">
    <template name="model:group">
        <listgroup open="${groupingInfo.open}" label="${each}" />
    </template>
    <template name="model">
        <listitem>....</listitem>
    </template>
    <template name="model:groupfoot">
        <listgroupfoot>....</listgroupfoot>
    </template>
</listbox>

```

- Note the `groupingInfo` is used to get the information of the grouping data. Please refer to GroupingInfo (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/GroupingInfo.html#>).

Version History

Version	Date	Content
6.0.0	July 2011	The template feature was introduced.
6.0.0	January 2012	The GroupingInfo statement was introduced.

Grid Template

Similar to Listbox, you can define a customer rendering with a template for a grid:

```
<grid model="${books}">
    <columns>
        <column label="ISBN" sort="auto"/>
        <column label="Name" sort="auto"/>
        <column label="Description"/>
    </columns>
    <template name="model">
        <row>
            <label value="${each.isbn}" />
            <label value="${each.name}" />
            <label value="${each.description}" />
        </row>
    </template>
</grid>
```

where `books` is assumed as an instance of `ListModel`^[7] that contains a list of the Book instances while each Book instance has at least three getter methods: `getIsbn`, `getName` and `getDescription`.

Notice that the template named `model` must be associated with the grid, i.e., it must be a direct child element of the `grid` element as shown above. A common mistake is to put it under the `rows` element. Remember the template is a ZUML fragment telling the grid how to render a row, and the template itself is not a component.

Template for GroupsModel

When used with `GroupsModel`^[1], grids will use the template called `model:grouping` for rendering the grouping object. If it is not defined, it will look for the template called `model` instead (i.e., the same template is used for rendering the grouping and non-grouping objects).

```
<grid mode="${fooGroupsModel}">
    <template name="model:group">
        <group open="${groupingInfo.open}">....</group>
    </template>
    <template name="model">
        <row>....</row>
    </template>
    <template name="model:groupfoot">
        <groupfoot>....</groupfoot>
    </template>
```

```
<grid>
```

- Note the *groupingInfo* is used to get the information of the grouping data. Please refer to GroupingInfo^[1].

Version History

Version	Date	Content
6.0.0	July 2011	The template feature was introduced.
6.0.0	January 2012	The GroupingInfo statement was introduced.

References

[1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ext/GroupingInfo.html#>

Tree Template

Similar to Listbox, you can also define a customer rendering with a template for a tree:

```
<tree model="${files}">
    <treetable>
        <treetablecol label="Path"/>
        <treetablecol label="Description"/>
    </treetable>
    <template name="model">
        <treetableitem context="menupopup">
            <treetablerow>
                <treetablecell label="${each.data.path}"/>
                <treetablecell label="${each.data.description}"/>
            </treetablerow>
        </treetableitem>
    </template>
</tree>
```

assume `files` is an instance of DefaultTreeModel^[2]. Notice since DefaultTreeModel^[2] is used, each references an instance of DefaultTreeNode^[3]. Thus, to retrieve the real data, use `DefaultTreeNode.getData()`^[1]

Version History

Version	Date	Content
6.0.0	July 2011	The template feature was introduced.

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeNode.html#getData\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/DefaultTreeNode.html#getData())

Combobox Template

Similar to Listbox, you can render a combobox with a template:

```
<combobox model="${infos}">
    <template name="model">
        <comboitem label="${each[0]} : ${each[1]}" />
    </template>
</combobox>
```

where we assume there is a list model (ListModel^[7]) called `infos` such as:

```
ListModel infos = new ListModelArray(
    new String[][] {
        {"Apple", "10kg"},
        {"Orange", "20kg"},
        {"Mango", "12kg"}
    });
}
```

Version History

Version	Date	Content
6.0.0	July 2011	The template feature was introduced.

Selectbox Template

Similar to Listbox, you can render a selectbox with a template. However, notice that, unlike other components, selectbox doesn't allow any child component, so you have to render each item as a string. For example,

```
<selectbox model="${users}" onSelect='alert(model.getData());'>
    <template name="model">
        Name is ${each}
    </template>
</selectbox>
```

where we assume there is a list model (ListModel^[7]) called `users` such as:

```
ListModelList model = new ListModelList(new String[] { "Tony", "Ryan",
    "Jumper", "Wing", "Sam" });
```

Version History

Version	Date	Content
6.0.0	November 2011	The selectbox component was introduced.
6.0.0	July 2011	The template feature was introduced.

Biglistbox Template

Similar to Listbox, you can render a biglistbox with a template. However, notice that, unlike other components, biglistbox doesn't allow any child component, so you have to render each item as a string. For example,

```
<biglistbox hflex="1" vflex="1" model="${data}" >
    <!-- Template example -->
    <template name="heads">
        <html><! [CDATA[
            <div class="images_${matrixInfo[0]}" title="x=${matrixInfo[0]},y=${matrixInfo[1]}>${each[matrixInfo[0]]}</div>
        ]]></html>
    </template>
    <template name="rows">
        <html><! [CDATA[
            <div class="images_${matrixInfo[0]}" title="x=${matrixInfo[0]},y=${matrixInfo[1]}>${each[matrixInfo[0]]}</div>
        ]]></html>
    </template>
</biglistbox>
```

As you can see, we utilize two attributes - `rowIndex` & `colIndex` from the `matrixInfo` object to receive the current index during template rendering phase.

where we assume there is a matrix model (FakerMatrixModel^[5]) called `data` such as:

```
FakerMatrixModel model = new FakerMatrixModel(100, 100);
```

Version History

Version	Date	Content
6.0.1	March 2012	The biglistbox component was introduced.

Chosenbox Template

Similar to Listbox, you can render a chosenbox with a template. However, notice that, unlike other components, chosenbox doesn't allow any child component, so you have to render each item as a string. For example,

```
<chosenbox model="${users}" onSelect='alert(model.getData());'>
    <template name="model">
        Name is ${each}
    </template>
</chosenbox>
```

where we assume there is a list model (ListModel^[7]) called `users` such as:

```
ListModelList model = new ListModelList(new String[] { "Tony", "Ryan",
"Jumper", "Wing", "Sam" });
```

Version History

Version	Date	Content
6.0.1	April 2012	The chosenbox component was introduced.
6.0.0	July 2011	The template feature was introduced.

Tabbox Template

- Available for ZK:
- CE** **PE** **EE**

The template used to control the rendering of each tab andtabpanel must be named `model:tab` and `model:tabpanel` and declared right inside the `tabbox` element. For example,

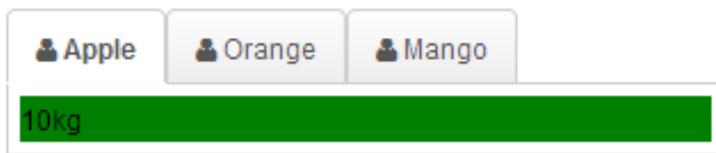
```
<div apply="foo.FruitProvider">
<tabbox id="mytab" model="${$composer.fruits}">
    <template name="model:tab">
        <tab iconSclass="z-icon-user">
            ${each[0]}
        </tab>
    </template>
    <template name="model:tabpanel">
        <tabpanel>
            <div style="background:green">
                ${each[1]}
            </div>
        </tabpanel>
    </template>
</tabbox>
</div>
```

The template's name is important because users are allowed to associate multiple templates to one component, and `tabbox`'s default renderer looks only for the template called `model:tab` and `model:tabpanel`.

When the template is rendered, a variable called `each` is assigned with the data being rendered. Thus, you could retrieve the information to render with EL expressions, such as `${each[0]}`, if it is an array, or `${each.name}`, if it is a bean with a getter called `name`.

In this example, we assume the `$composer.fruits` expression returns a two-dimensional array^[1], and is provided by the `foo.FruitProvider` composer such as follows^[2].

```
<zscript><! [CDATA[
public class FruitProvider extends
org.zkoss.zk.ui.select.SelectorComposer {
    public ListModelArray fruits = new ListModelArray(
        new String[][] {
            {"Apple", "10kg"},
            {"Orange", "20kg"},
            {"Mango", "12kg"}
        });
    public ListModelArray getFruits() {
        return fruits;
    }
}
]]></zscript>
```



- [1] Of course, it can be anything you like. Just make sure it matches the EL expressions specified in the template.
 [2] Here we use UNIQ-javadoc-0-736dd3f9533a2ca2-QINU for simplicity. There are several ways to implement a composer, such as wiring a Spring-managed bean. For more information, please refer to the Composer section

Component's Value

By default, the data used to render a component will be stored to the component's value property automatically. For tab, it is Tab.setValue(T) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Tab.html#setValue\(T\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Tab.html#setValue(T))). Thus, you retrieve it back easily by invoking Tab.getValue() ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Tab.html#getValue\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Tab.html#getValue())).

Of course, if you prefer to store other values, you can simply specify `value="${whatever}"` to the tab element in the template.

The forEachStatus Variable

There is a variable called forEachStatus providing the information of the iteration. It is an instance of ForEachStatus (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ForEachStatus.html#>). For example, you could retrieve the iteration's index by use of `${forEachStatus.index}`.

Lifecycle and the arg Variable

When using the template, it is important to remember that the template is rendered on demand. It means the template can be rendered very late, after the page is rendered, after the user scrolls down to make an item visible, and so on. Thus, in the template, you *cannot* reference anything that is available only in the page rendering phase. For example, you can't reference the arg variable in a template:

```
<div apply="foo.FruitProvider">
<tabbox id="mytab" model="${$composer.fruits}">
  <template name="model:tab">
    <tab label="${arg.foo}" /> <!-- Wrong! it is always empty -->
  </template>
  <template name="model:tabpanel">
    <tabpanel>
      ${each[1]}
    </tabpanel>
  </template>
</tabbox>
</div>
```

To work around, you have to store the value in, say, component's custom attributes (Component.getAttributes() ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Component.html#getAttributes\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Component.html#getAttributes()))). For example,

```
<div apply="foo.FruitProvider">
<tabbox id="mytab" model="${$composer.fruits}">
  <custom-attributes foo="${arg.foo}" /><!-- store it for later use -->
  <template name="model:tab">
```

```

<tab label="${foo}" /> <!-- Correct! Use the stored copy. -->
</template>
<template name="model:tabpanel">
    <tabpanel>
        <div>
            ${each[1]}
        </div>
    </tabpanel>
</template>
</tabbox>
</div>

```

Version History

Version	Date	Content
7.0.0	November 2013	Tabbox support ListModel (http://tracker.zkoss.org/browse/ZK-2002)

Organigram Template

- Available for ZK:
- CE PE EE

Similar to Listbox, you can also define a customer rendering with a template for an organigram:

```

<zscript><![CDATA[
    DefaultTreeNode root = new DefaultTreeNode(null, new
DefaultTreeNode[] {
        new DefaultTreeNode("Item1", new DefaultTreeNode[] {
            new DefaultTreeNode("Item2"), new
DefaultTreeNode("Item3"), new DefaultTreeNode("Item4")
        })
    });
    DefaultTreeModel model = new DefaultTreeModel(root);
    model.addOpenPath(new int[]{0});
]]></zscript>
<organigram model="${model}">
    <template name="model">
        <orgitem>
            <orgnode label="${each.data}" />
        </orgitem>
    </template>
</organigram>

```

Searchbox Template

Similar to Listbox, you can render a searchbox with a template. However, notice that, unlike other components, searchbox doesn't allow any child component, so you have to render each item as a string, a <label> or a <html>. For example,

```
<searchbox model="${users}">
    <template name="model">
        Name is ${each}
    </template>
</searchbox>

<searchbox model="${users}">
    <template name="model">
        <label value="Name is ${each}" />
    </template>
</searchbox>

<searchbox model="${users}">
    <template name="model">
        <html><! [CDATA[
            Name is ${each}
        ]]></html>
    </template>
</searchbox>
```

where we assume there is a list model (ListModel^[7]) called `users` such as:

```
ListModelList model = new ListModelList(new String[] { "Tony", "Ryan",
"Jumper", "Wing", "Sam" });
```

Version History

Version	Date	Content
9.0.0	September 2019	ZK-4380 ^[1] : Provide a Searchbox component

References

[1] <https://tracker.zkoss.org/browse/ZK-4380>

Renderer

A **renderer** is a Java class that produces child items based on a data model. It renders your data into View. There are 2 kinds of renderers:

1. . Render child components: like ListitemRenderer^[1], RowRenderer^[2]
2. . Render HTML snippets: ItemRenderer^[3].

The implementation of a renderer depends on the component. For example, the display of Listbox^[3] can be customized by an implementation of ListitemRenderer^[1], and Grid^[3] by RowRenderer^[2].

If you prefer to define the rendering of each item in the ZUML document, you can use templates instead.

References

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

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

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

Listbox Renderer

Here we describe how to implement a custom renderer for a listbox (ListitemRenderer^[1]). For the concepts about component, model and renderer, please refer to the Model-driven Display section.

When a listbox (Listbox^[3]) is assigned with a model, a default renderer is assigned too. The default renderer will assume that each list item has only one column, and it converts the data into a string directly^[1]. If you want to display multiple columns or retrieve a particular field of the data, you have to implement ListitemRenderer^[1] to handle the rendering.

For example,

```
public class MyRenderer implements ListitemRenderer{
    public void render(Listitem listitem, Object data, int index) {
        Listcell cell = new Listcell();
        listitem.appendChild(cell);
        if (data instanceof String[]) {
            cell.appendChild(new
Label(((String[])data)[0].toString()));
        } else if (data instanceof String) {
            cell.appendChild(new Label(data.toString()));
        } else {
            cell.appendChild(new
Label("UNKNOW:" + data.toString()));
        }
    }
}
```

[1] If the listbox is assigned a template called `model`, then the template will be used to render the listbox. For more information, please refer to the Listbox Template section.

Version History

Version	Date	Content
6.0.0	February 2012	The index argument was introduced.

Grid Renderer

When a Grid ^[3] is assigned with a model, a default renderer is assigned too (see the Model-driven Display). The default renderer will assume that each row has only one column, and it converts the data into a string directly^[1]. If you want to display multiple columns or retrieve a particular field of the data, you have to implement RowRenderer ^[2] to handle the rendering and assign it by `setRowRenderer()` ^[2].

For example,

```
public class FoodGroupRenderer implements RowRenderer,
java.io.Serializable {
    public void render(Row row, Object obj, int index) {
        if (row instanceof Group) {
            row.appendChild(new Label(obj.toString()));
        } else {
            User user = (User) obj;
            row.appendChild(new Label(user.getName()));
            row.appendChild(new Label(user.getDescription()));
            row.appendChild(new Label(user.getDomain()));
        }
    }
}
```

[1] If the grid is assigned a template called `model`, then the template will be used to render the grid. For more information, please refer to the Grid Template section.

[2] <https://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Grid.html#setRowRenderer-org.zkoss.zul.RowRenderer->

Version History

Version	Date	Content
6.0.0	February 2012	The index argument was introduced.

Tree Renderer

When a tree (Tree^[1]) is assigned with a model, a default renderer is assigned too^[1]. The default renderer will assume that each tree item has only one column, and it converts the data into a string directly^[2]. If you want to display multiple columns or retrieve a particular field of the data, you have to implement TreeitemRenderer^[3] to handle the rendering.

For example,

```
public class HostTreeRenderer implements TreeitemRenderer {
    public void render(Treeitem treeitem, Object data, int index)
throws Exception {
    Treerow row = treeitem.getTreerow();
    if (row == null) { // tree row not create yet.
        row = new Treerow();
        treeitem.appendChild(row);
    }
    if (data instanceof HostTreeModel.FakeGroup) {
        treeitem.getTreerow().appendChild(new
Treecell(((HostTreeModel.FakeGroup) data).getName()));
    } else if (data instanceof HostTreeModel.FakeHost) {
        treeitem.getTreerow().appendChild(new
Treecell(((HostTreeModel.FakeHost) data).getName()));
    } else if (data instanceof HostTreeModel.FakeProcess) {
        treeitem.getTreerow().appendChild(new
Treecell(((HostTreeModel.FakeProcess) data).getName()));
    }
}
```

[1] For the concept about component, model and renderer, please refer to the Model-driven Display section.

[2] If the tree is assigned a template called `model`, then the template will be used to render the tree. For more information, please refer to the Tree Template section.

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

Version History

Version	Date	Content
6.0.0	February 2012	The index argument was introduced.

Combobox Renderer

When a combobox (Combobox^[2]) is assigned with a model, a default renderer is assigned too^[1]. The default renderer will assume that the combobox displays the data as a string^[2]. If you want to display more sophisticated information or retrieve a particular field of the data, you have to implement ComboitemRenderer^[3] to handle the rendering.

For example,

```
public class MyRenderer implements ComboitemRenderer {  
    public void render(Comboitem item, Object data, int index) throws  
        Exception {  
        item.setLabel(((User) data).getName());  
        item.setDescription(((User) data).getDescription());  
    }  
}
```

[1] For the concept about component, model and renderer, please refer to the Model-driven Display section.

[2] If the tree is assigned a template called `model`, then the template will be used to render the tree. For more information, please refer to the Tree Template section.

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

Version History

Version	Date	Content
6.0.0	February 2012	The index argument was introduced.

Tabbox Renderer

- Available for ZK:
- **CE** **PE** **EE**

Here we describe how to implement a custom renderer for a tabbox (TabboxRenderer^[1]). For the concepts about component, model and renderer, please refer to the Model-driven Display section.

When a tabbox (Tabbox^[1]) is assigned with a model, a default renderer is assigned too. The default renderer will assume that each tab has only onetabpanel, and it converts the data into a string directly^[2]. If you want to display a richtabpanel or retrieve a particular field of the data, you have to implement TabboxRenderer^[1] to handle the rendering.

For example,

```
public class MyRenderer implements TabboxRenderer{
    public void renderTab(Tab tab, Object data, int index) {
        tab.setLabel("New -- " + data);
    }
    public void renderTabpanel(Tabpaneltabpanel, Object data, int
index) {
       tabpanel.appendChild(new Label("New -- " + data));
    }
}
```

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

[2] If the tabbox is assigned a template called `model:tab` and `model:tabpanel`, then the template will be used to render the tabbox.

For more information, please refer to the Tabbox Template section.

Organigram Renderer

- Available for ZK:



When an organigram (Organigram^[1]) is assigned with a model, a default renderer is assigned too^[2]. The default renderer will assume that each Orgitem has one Orgnode, and it converts the data into a string directly^[3]. If you want to change render style or retrieve a particular field of the data, you have to implement OrgitemRenderer^[4] to handle the rendering.

For example,

```
public class MyRenderer implements OrgitemRenderer {  
    public void render(Orgitem orgitem, Object data, int index)  
throws Exception {  
    final Button button = new  
Button(Objects.toString(data));  
    button.addEventListener(Events.ON_CLICK, new  
EventListener<Event>() {  
        public void onEvent(Event event) {  
  
Clients.showNotification(button.getLabel());  
        }  
    } );  
  
    Orgnode orgnode = new Orgnode();  
    button.setParent(orgnode);  
    orgnode.setParent(orgitem);  
}
```

[1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkmax/zul/Organigram.html#>

[2] For the concept about component, model and renderer, please refer to the Model-driven Display section.

[3] If the Organigram is assigned a template called `model`, then the template will be used to render the Organigram. For more information, please refer to the Organigram Template section.

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

Biglistbox Renderer

The implementation of a custom renderer for a Biglistbox (MatrixRenderer^[1]) is straightforward^[2]:

```
public class DataRenderer implements org.zkoss.zkmax.zul.MatrixRenderer<List<String>>
{
    @Override
    public String renderCell(Component owner, List<String> data,
                           int rowIndex, int colIndex) throws Exception {
        String d = data.get(colIndex);
        d = d.replace("ZK", "<span class='red' title='ZK'>ZK</span>");
        d = d.replace("Hello", "<span class='blue' title='Hello'>Hello</span>");
        return "<div class='images_" + (colIndex%28) + "' title='x=" +
               colIndex + ",y=" + rowIndex + "'>" + d + "</div>";
    }

    @Override
    public String renderHeader(Component owner, List<String> data,
                           int rowIndex, int colIndex) throws Exception {
        return "<div class='images_" + (colIndex % 28) + "' title='"
               + images[colIndex % 28] + "'>" + data.get(colIndex)
               + "</div>";
    }
}
```

Then, if we have a list model (MatrixModel^[1]) called data, and an instance of a custom renderer called dataRenderer, then we can put them together in a ZUML document as follows:

```
<biglistbox model="${data}" matrixRenderer="${dataRenderer}" />
```

[1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkmax/zul/MatrixRenderer.html#>

[2] For the concept about component, model and renderer, please refer to the Model-driven Display section.

Version History

Version	Date	Content
6.0.1	March 2012	Biglistbox was introduced.

Item Renderer

For those components that have no child components e.g. Chosenbox, Selectbox, Cascader, Searchbox, their built-in renderers will directly wrap your data into an HTML snippet. Hence if your model data contains those characters that need to be escaped ^[1] like < > &, you must escape them. You can call ZK's XMLs.escapeXML() ^[2] or Strings.escapeJavaScript() ^[3].

Implementing your own (ItemRenderer ^[3]) can customize how a component renders data in a browser without javascript. (For the concepts about component, model, and renderer, please refer to the Model-driven Display section). Notice that `ItemRenderer` should return an HTML snippet that is different from `ListitemRenderer` ^[1] that creates components.

Render an item with a tooltip

```
public class TooltipRenderer implements ItemRenderer {  
    @Override  
    public String render(Component owner, Object data, int index)  
    throws Exception {  
        return String.format("<span title=\"%s\" style=\"width: 100%;display: inline-block;\">" + "%s" + "</span>", data, data);  
    }  
}
```

Usage

Assume we have a tree model (TreeModel ^[4]) called `district`, and an instance of a custom renderer called `districtRenderer`, we can put them together in a ZUML document as follows:

```
<cascader model="${district}" itemRenderer="${districtRenderer}" />
```

Specify FQCN at `itemRenderer`

```
<cascader model="${model}" itemRenderer="org.zkoss.reference.component.input.TooltipRenderer" />
```

References

- [1] <https://www.w3.org/International/questions/qa-escapes#use>
- [2] <https://www.zkoss.org/javadoc/latest/zk/org/zkoss/xml/XMLs.html#escapeXML-java.lang.String->
- [3] <https://www.zkoss.org/javadoc/latest/zk/org/zkoss/lang/Strings.html#escapeJavaScript-java.lang.String->

Stateless Components

Overview

Classic ZK overview

Classic ZK components are stored in the HttpSession on the server side. Based on the Java EE Web application specification, a session is an object that exists in memory on the server side. When a web browser sends a request to the server, it also sends a session identifier. This is commonly done with a JSESSIONID cookie or through a URL parameter.

Based on this identifier, the Web Server will retrieve the session object from memory and make it available while processing the response to the user's request. This allows the server-side application to store various objects semi-permanently in the server's memory until the session is destroyed.

In the classic use case, the page generated by ZK exists in the session. When the user triggers an action or changes the state of a component on the client side, a request is sent to the server and updates the "main state" of the components that exist in the session. In this workflow, the server-side state is authoritative, and the client state is synchronized based on the server-side state.

Stateless components overview

The stateless components are not stored in the server's memory. They are transient objects which only exist during the user's request and response cycle.

In this case, the authoritative state is located on the client side. Updates performed with Stateless components are done on a "per action" basis.

Each action has a set of "Action parameters", which indicates what data needs to be retrieved from the client in order to process a given action. This way, no server-side record of the page's state is required. When the user triggers an action, the client-side engine will fetch the properties and values requested by the action and send all relevant data in the request.

The server-side action will then process any relevant update based on that data and the event triggered by the user. At the end of that processing, the action may modify the client's state and send a response that will write the new state on the client side.

During this process, the action only used the information retrieved by the action parameters. As such, it did not need to access any data located on the server side. s.

Differences Between Classic ZK Components and Stateless Components

Component State Management

- **Classic:** Stored in the HttpSession on the server side, maintaining states in the server's memory for semi-permanent object storage.
- **Stateless:** Not stored in server memory, existing only during the user's request-response cycle. The state is stored in a browser.

State Authority

- **Classic:** Server-side authoritative state, with client-side state synchronized based on the server-side state.
- **Stateless:** Client-side authoritative state, with updates performed on a per-action basis.

Data Processing and Memory Usage

- **Classic:** Maintain consistent state across user interactions, requiring more server memory for state retention.
- **Stateless:** Lower memory footprint due to lack of need for server-side state record between transactions.

Use Cases and Suitability

- **Classic:** Suitable for applications requiring server-side permanence, and traditional web infrastructure.
- **Stateless:** Ideal for distributed infrastructures with session replication, applications with limited server-side memory, and services needing high scalability, like cloud-native applications.

Advantages: Scalability, cloud readiness, and resource efficiency

The Stateless Components offer several advantages:

- **Scalability:** They adapt easily to varying workloads, making them ideal for cloud-native applications.
- **Cloud Readiness:** Simplifies deployment and management of applications in cloud environments.
- **Resource Efficiency:** Reduces server memory usage as there is no need to maintain component state on the server.

Limitations

WebSocket Not Supported

Since WebSocket is a stateful connection, it's not supported for StatelessRichlet.

Cannot Work with Stateful Components

Stateless and classic components cannot be used on one page together. The basic architecture for these components is different. Using them in one application is also not supported.

But a combination of Iframes and ZK's embedded features can effectively bring stateless and classic components together. For example, the main page could have an Iframe targeting a ZK stateless service. In contrast, another panel with more direct interaction could contain a classic ZK page added by ZK Embedded API.

FAQ

Are Stateless components an upgrade to classic ZK components

No. Stateless components are a branching technology from the classic ZK components. They use the same client-side code (the JavaScript Widgets and associated HTML and CSS code), in addition to a completely different communication and update layer.

How to handle use cases that require server-side permanence

While the Stateless components only exist during requests and responses, you may need to store data that the users should not be able to modify or data that should not be publicly accessible.

In this situation, some sort of shared permanence layer is necessary for the application's function. For this purpose, a good option is to use a database layer. In the stateless components demo application, the content of the user's shopping basket is stored in a database. Since the data layer is decoupled from the web container, it can be accessed by any number of instances. Databases already provide replication and high availability features, which makes them a great candidate to act as the shared permanence layer.

I have an existing ZK application...

... using classic ZK, do I need to migrate my code to use Stateless components instead?

It depends on your requirements. We do not see the Stateless component as "the next step" in ZK innovation. Instead, we see them as a convenient tool that can be used if they are relevant to a specific application's architecture requirements.

As a rule of thumb, consider the following.

I should use stateless components if:

- I'm developing an application for a heavily distributed infrastructure in which session replication between nodes is difficult to achieve (cloud hosting, high availability with multiple regional clusters, etc.)
- I'm developing smaller-scale services that do not need to know about each other to perform their functions.
- I do not have access to sticky sessions, which causes the active node for a given user to change with each request
- I am concerned about the memory footprint, I would like to minimize the memory used on the server side.

I should use classic ZK if:

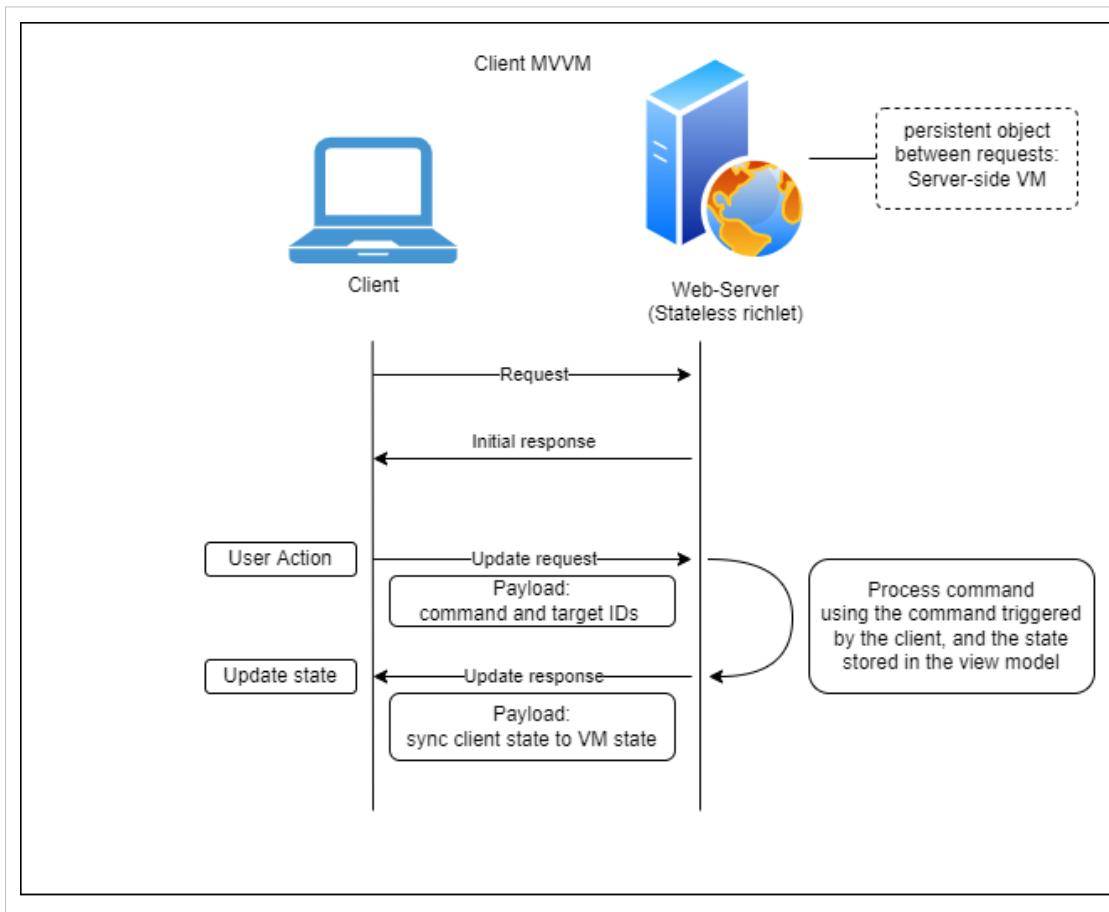
- I already have a fully functional application, which matches my infrastructure needs
- My infrastructure is a classic "Database, webserver (or webserver cluster), gateway/reverse proxy"
- I need to store multiple data, states, and other semi-permanent objects between requests
- My application has a lot of interdependent systems that rely on server-side communication

Comparing and contrasting the Stateless workflow, and the new ZK 10 Client-MVVM features

Both of these new features' goal is to improve the resource footprint of a given page on the server side. There is a similar concept between both features: removing server-side component instances.

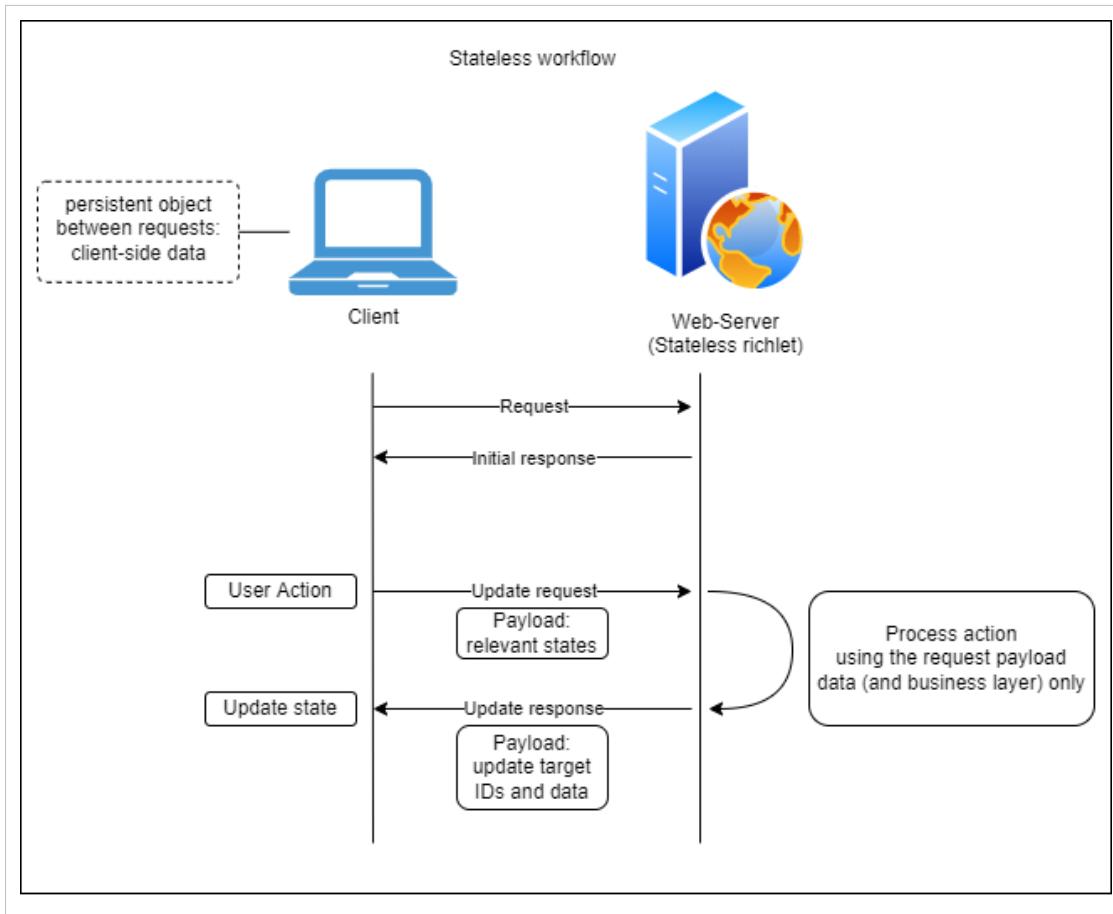
In the case of Client-side MVVM, the ClientBindComposer will keep track of the bindings and commands but not create corresponding ZK components (the textboxes, grids, buttons, etc) in the server memory.

The view model being already decoupled from the view doesn't need to know that the command comes from a server-side Java instance of Textbox, or if it was sent by the client, and forwarded directly by the ClientBindComposer. In the opposite direction, the client-side Textbox JavaScript object doesn't know if the update returned by the response is generated by a server-side Java Textbox object or forwarded directly from the view model by the ClientBindComposer.



In the case of stateless components, the Java-side objects are also removed. Instead of maintaining a state in the view model, and using the ClientBindComposer to forward that state to the client-side objects, the stateless workflow uses a list of inputs and outputs for each user action in a pattern that is closer to MVC.

When a user triggers an action on the client side, all the state data required to process that action is sent together with the request. As a result, there is no state at all located in server memory.



Can stateless components work with server push?

Stateless components in ZK, due to their architectural nature, present certain limitations when it comes to working with server push mechanisms. Unlike classic ZK components that maintain their state on the server, stateless components are designed to be transient, with their state predominantly managed on the client side. This design choice significantly reduces server-side memory footprint but also impacts how stateless components interact with server push technologies.

Server push, typically used to update client-side UI elements from the server in real time, relies on a persistent connection between the client and the server. In the context of stateless components, it's not supported to use server push.

Building Stateless UI

Setting up

To set up stateless components in a ZK 10 application, you need to include the stateless components module and define a Dispatcher Richlet Filter in your `WEB-INF/web.xml` file.

Including Required Jar

```
dependencies {  
    implementation "org.zkoss.zk:stateless:${zkVersion}"  
    ...  
}
```

Dispatcher Richlet Filter

```
<filter>  
    <filter-name>DispatcherRichletFilter</filter-name>  
    <filter-class>org.zkoss.stateless.ui.http.DispatcherRichletFilter</filter-class>  
    <init-param>  
        <param-name>basePackages</param-name>  
        <param-value><!-- your base packages --></param-value>  
    </init-param>  
</filter>  
<filter-mapping>  
    <filter-name>DispatcherRichletFilter</filter-name>  
    <url-pattern>/*</url-pattern>  
</filter-mapping>
```

Example Application

We will use the simple shopping cart application as an example to introduce the basic features of stateless components (download the shopping cart example project ^[1]):

Building UI with Richlet

Building user interfaces (UI) with stateless components requires creating a StatelessRichlet and mapping a URL to that richlet.

URL Mapping

We use `@RichletMapping` to compose a URL. When users visit that URL, ZK will invoke the corresponding method.

For example below, the `index()` URL will be `<protocal>://<host name: port> /shoppingCart`.

```
@RichletMapping("/shoppingCart")  
public class DemoRichlet implements StatelessRichlet {  
    @RichletMapping("")
```

```
public List<IComponent> index() {
    //return ...
}
```

Class-Level Mapping

At this level, `@RichletMapping` defines the base path for all methods in a `StatelessRichlet`. For example, assigning `@RichletMapping("/shoppingCart")` to the `DemoRichlet` class sets a foundational path for all UI components it manages.

Method-Level Mapping

Within the `StatelessRichlet`, each method can specify further URL mapping. By applying `@RichletMapping("")` to a method, the specified path appends to the class-level path.

Composing the UI with Stateless Components

Before ZK 10, ZK components are stateful, meaning that the server holds the state. Starting from ZK 10, we provide a set of stateless components as **Immutable objects**. Immutable objects are constructed once and can not be changed after they are constructed. After Immutable objects are rendered, they will be destroyed. Since the stateless component states will not be saved on the server, they consume less memory.

With stateless components, ZK offers a streamlined, fluent API for building user interfaces.

- Every classic component has a corresponding stateless version, identified by an "I" prefix, denoting "immutable."
- Stateless components employ a builder pattern, using methods like `of()` for initializing properties and `withSclass()` for setting classes.

Here's a comparison of UI composition between classic and stateless components:

Classic Component in ZK 9

```
Button button = new Button("add items");
button.setSclass("add-items");
```

Equivalent Stateless Component in ZK 10

```
IButton.of("add items")
.withSclass("add-items");
```

Therefor, for the method with URL mapping, we should return a list of components like:

```
@RichletMapping("")
public List<IComponent> index() {
    return asList(
        IStyle.ofSrc(DEMO_CSS),
        ILayout.of(
            renderShoppingCart(),
            Boilerplate.ORDER_TEMPLATE
        )
    );
}
```

Event Wiring

To wire an action handler method for an event, you need to call `withAction(ActionHandler action)` with a **public method** reference:

```
IButton.of("add item +")
    .withSclass("add-items")
    .withAction(ActionType.onClick(this::addItem))
```

- Line 3: it means register `addItem()` as an action handler for `onClick` event on `IButton`. `ActionType` supports all types of component events.

In stateless components, we use the term **action handler**, which distinctly separates it from the event listener associated with classic components.

Hence, when a user clicks the button above, ZK will invoke `addItem()` declared in the Richlet.

Obtain Component State

Since a server no longer holds a component's state (it's on the client side), we provide `@ActionVariable` to access a UI component's state sent from the client. When ZK invokes an action handler for an event, it will pass the corresponding parameters you specified.

- `@ActionVariable(targetId = ActionTarget.SELF, field = "id")` retrieves the value from the field of a component with the `targetId` on the client.
- `ActionTarget.SELF` represents the component associated with the event which is a button. Please see `ActionTarget[2]` for other targets.

```
public void addItem(@ActionVariable(targetId = ActionTarget.SELF,
field = "id") String orderId) {
```

- in this case, we will get a button's id

Get User Input

If you register an action handler on an input component like `ICombobox`:

```
ICombobox.of(initProductSize)
    .withReadonly(true)
    .withAction(ActionType.onChange(this::doSizeChange))
```

Declare `InputData` in the handler's signature, ZK will pass user input to you:

```
public void doSizeChange(InputData data,
                        @ActionVariable(targetId =
ActionTarget.PARENT, field = "id") String uuid) {
    String value = data.getValue();
}
```

Update Component State

ZK provides various APIs on `UiAgent` to update a component's state. You need to call its method in an action handler method to implement your UI logic. Then those commands to update component states will be sent to the client after executing the method.

Locator

When you manipulate stateless components with `UiAgent` API, you need to pass a `Locator`. Why? Because your Richlet doesn't have any reference to a stateless component on the server side, we don't have any setter to call. Hence, we need to tell ZK client engine the target component we want to manipulate by describing its location with `Locator`.

By Component ID

```
Locator.ofId("myId")
```

Self

If you declare `Self` on an action handler method signature, ZK will pass the event target component's Locator called `Self`. For example, if I wire the method below with the spinner above for quantity change:

```
public void doQuantityChange(Self self, ...)
```

- `Self` is the Locator of the spinner.

By Relative Position

If you have a Locator, you can find another component based on it like

```
self.nextSibling();
self.closest() //find its parent component
self.firstChild()
```

Remove

The following code removes a component specified by `Locator`.

```
public void doDelete(@ActionVariable(targetId =
ActionTarget.PARENT, field = "id") String id) {
    ...
    UiAgent.getCurrent().remove(Locator.ofId(id));
}
```

Add Child Components

```
public void addItem(@ActionVariable(targetId = ActionTarget.SELF, field
= "id") String id) {
    UiAgent.getCurrent()
        .appendChild(Locator.ofId(SHOPPING_CART_ROWS),
                    renderShoppingCartOneItem(parseOrderId(id)));
}
```

Change Component's Property

```
UiAgent.getCurrent()
    .smartUpdate(Helper.getPriceLocator(self),
        new ILabel.Updater().value(String.valueOf(price)))
```

- It changes a label's value with a price

References

- [1] <https://github.com/zkoss-demo/zk10-shopping-cart-demo>
[2] <https://www.zkoss.org/javadoc/latest/zk/org/zkoss/stateless/action/ActionTarget.html>

Annotations

An annotation is a special form of syntactic metadata that can be added to components. The definitions, properties and components themselves may be annotated. The annotations can be retrieved at the run time.

The annotations have no direct effect on the operation of the components. Rather, they are mainly used for UI designers to annotate metadata, such that it controls how a tool or a utility shall do at run-time. The content and meanings of annotations totally depend on the tools or the utilities the developer uses. For example, ZK Bind examines annotations to know how to load and store the value of a component.

Annotate in ZUML

Annotations can be applied to the declarations of components and properties in ZUML pages.

Annotate Properties

To annotate a property, you could specify an annotation expression as the value of the property. In other words, if the value of the property is an annotation expression, it is considered as an annotation for the property, rather than a value to be assigned.

The format of an annotation expression:

```
@annotation-name ()
@annotation-name ( attr-name1 = attr-value1, attr-name2 = attr-value2 )
@annotation-name ( attr-name1 = { attr-value1-1, attr-value1-2 }, attr-name2 = attr-value2 )
```

As shown, an annotation consists of an annotation name and any number of attributes, and an attribute consists of an attribute name and an attribute value. The name of an annotation must start with a letter ('a' - 'z' or 'A' - 'Z'), an underscore ('_'), or a dollar sign ('\$').

If an attribute has multiple values, these values have to be enclosed with the curly braces (as shown in the third format).

For example,

```
<listitem label="@bind(datasource='author',value='selected')"/>
```

where an annotation called bind is annotated to the label property, and the bind annotation has two attributes: datasource and value.

If the attribute name is not specified, the name is assumed to be `value`. For example, the following two statements are equivalent:

```
<textbox value="@bind(vm.p1.firstName)"/>
<textbox value="@bind(value=vm.p1.firstName)"/>
```

Here is a more complex example.

```
<textbox value="@save(vm.person.firstName, before=['cmd1', 'cmd2'])"/>
```

where it annotates the `value` property with an annotation named `save`, and the annotation has two attributes: `value` and `before`. The value of the `before` attribute is a two-element array: '`cmd1`' and '`cmd2`'. Notice that the quotations, '`'` and `"`, will be preserved, so they will be retrieved exactly the same as they are specified in the ZUML document.

To annotate the same property with multiple annotations, you could specify them one-by-one and separate them with a space, as shown below.

```
<textbox value="@bind(vm.value1) @validator('validator1') errorMessage=@bind(vm.lastMessage1) />
```

In addition, you could annotate with multiple annotations that have the same name. For example,

```
<textbox value="@bind(vm.first) @bind(vm.second)"/>
```

where two annotations are annotated to the `value` property.

Annotate Components

To annotate a component, you could specify an annotation expression in a specific attribute called `self` as shown below.

```
<label self="@title(value='Hello World')"/>
```

where `self` is a keyword to denote the annotation which is used to annotate the component declaration, rather than any property.

The annotation Namespace

ZK Loader detects the annotation automatically. However, it may not be what you expect. Here we discuss how to resolve these conflicts.

Specify both value and annotation

If you'd like to specify both the value and the annotations of a given property, you could specify a namespace called `annotation` to distinguish them. For example,

```
<textbox value="a property's value" a:value="@save(vm.user)" xmlns:a="annotation"/>
```

Then, the `textbox`'s `value` property will be assigned with a value, "`a property's value`", and an annotation, `@save(vm.user)`.

Specify a value that looks like an annotation

If the value of a property looks like an annotation, you could specify a namespace other than annotation to tell ZK Loader not to interpret it as an annotation. For example,

```
<textbox u:value="@value()" xmlns:u="zul"/>
```

Then, `@value()` will be considered as a value rather than an annotation, and assigned to the textbox's value property directly.

Version History

Version	Date	Content
6.0.0	December 2011	The new syntax was introduced. For ZK 5's syntax, please refer to ZK 5's Developer's Reference. Though not recommended, it is OK to use ZK 5's syntax in ZK 6.

Annotate in Java

You could annotate a component or a property in Java by the use of `java.lang.String`, `java.util.Map`) `ComponentCtrl.addAnnotation(java.lang.String, java.lang.String, java.util.Map)`^[1].

For example,

```
Listbox listbox = new Listbox();
listbox.addAnnotation(null, "foo", null); //null in the first argument
means to annotate listbox
Label label = new Label();
label.addAnnotation("value", "fun", null); //annotate the value
property of label
```

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/ComponentCtrl.html#addAnnotation\(java.lang.String,%20java.lang.String,%20java.util.Map\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/ComponentCtrl.html#addAnnotation(java.lang.String,%20java.lang.String,%20java.util.Map)).

Retrieve Annotations

The annotations can be retrieved back at the run-time. They are designed to be used by tools or utilities, such as the data-binding manager, rather than applications. In other words, applications annotate a ZUML page to tell the tools how to handle components for a particular purpose.

The following is an example to dump all annotations of a component:

```
void dump(StringBuffer sb, Component comp) {
    ComponentCtrl compCtrl = (ComponentCtrl)comp;
    sb.append(comp.getId()).append(": ")
        .append(compCtrl.getAnnotations(null)).append('\n');

    for (String prop: compCtrl.getAnnotatedProperties()) {
        sb.append(" with ").append(prop).append(": ")
            .append(compCtrl.getAnnotations(prop)).append('\n');
    }
}
```

Annotate Component Definitions

In addition to annotating a component or its properties, you could annotate a component definition, such that all its instances will have the annotations.

To annotate a component definition, you have to specify the annotations in a language definition. For example, we could extend the definition of `bandbox` to add annotations. Please refer to ZK Component Reference/Annotation/Data Binding for detail.

```
<component>
    <component-name>bandbox</component-name>
    <extends>bandbox</extends>
    <annotation>
        <annotation-name>ZKBIND</annotation-name>
        <property-name>value</property-name>
        <attribute>
            <attribute-name>ACCESS</attribute-name>
            <attribute-value>both</attribute-value>
        </attribute>
        <attribute>
            <attribute-name>SAVE_EVENT</attribute-name>
            <attribute-value>onChange</attribute-value>
        </attribute>
        <attribute>
            <attribute-name>LOAD_REPLACEMENT</attribute-name>
            <attribute-value>rawValue</attribute-value>
        </attribute>
        <attribute>
            <attribute-name>LOAD_TYPE</attribute-name>
        </attribute>
    </annotation>

```

```
        <attribute-value>java.lang.String</attribute-value>
    </attribute>
</annotation>
<annotation>
    <annotation-name>ZKBIND</annotation-name>
    <property-name>open</property-name>
    <attribute>
        <attribute-name>ACCESS</attribute-name>
        <attribute-value>both</attribute-value>
    </attribute>
    <attribute>
        <attribute-name>SAVE_EVENT</attribute-name>
        <attribute-value>onOpen</attribute-value>
    </attribute>
</annotation>
</component>
```

UI Patterns

This section describes feature-specific UI handling topics. For introductory concepts, please refer to the UI Composing section. For detailed information of individual components, please refer to ZK Component Reference.

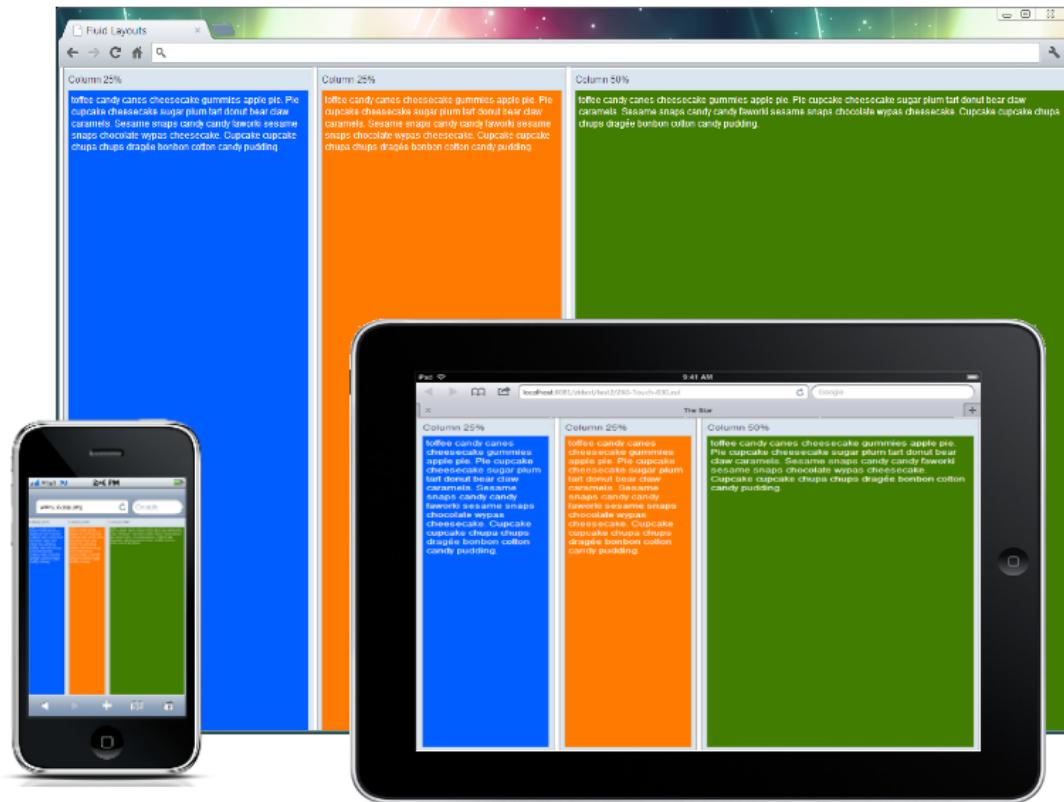
Responsive Design

The Responsive Design ^[1] in ZK separates the following three sections, for more features in tablet devices, please refer to Component Reference.

Fluid Layouts

You can adjust the component size using either *vflex* or *hflex* instead of giving components a fixed height and/or width in pixels.

For example,



```

<hlayout vflex="1">
    <window title="Column 25%" vflex="1" hflex="1" sclass="column1" border="normal">
        toffee candy canes cheesecake gummies apple pie. Pie
        cupcake cheesecake sugar plum tart donut
        bear claw caramels. Sesame snaps candy candy faworki
        sesame snaps chocolate wypas cheesecake.
        Cupcake cupcake chupa chups dragée bonbon cotton
        candy pudding.
    </window>
    <window title="Column 25%" vflex="1" hflex="1" sclass="column2" border="normal">
        toffee candy canes cheesecake gummies apple pie. Pie
        cupcake cheesecake sugar plum tart donut
        bear claw caramels. Sesame snaps candy candy faworki
        sesame snaps chocolate wypas cheesecake.
        Cupcake cupcake chupa chups dragée bonbon cotton
        candy pudding.
    </window>
    <window title="Column 50%" vflex="1" hflex="2" sclass="column3" border="normal">
        toffee candy canes cheesecake gummies apple pie. Pie
        cupcake cheesecake sugar plum tart donut bear claw
        caramels. Sesame snaps candy candy faworki
        sesame snaps chocolate wypas cheesecake. Cupcake cupcake chups
        dragée bonbon cotton candy pudding.
    </window>
</hlayout>

```

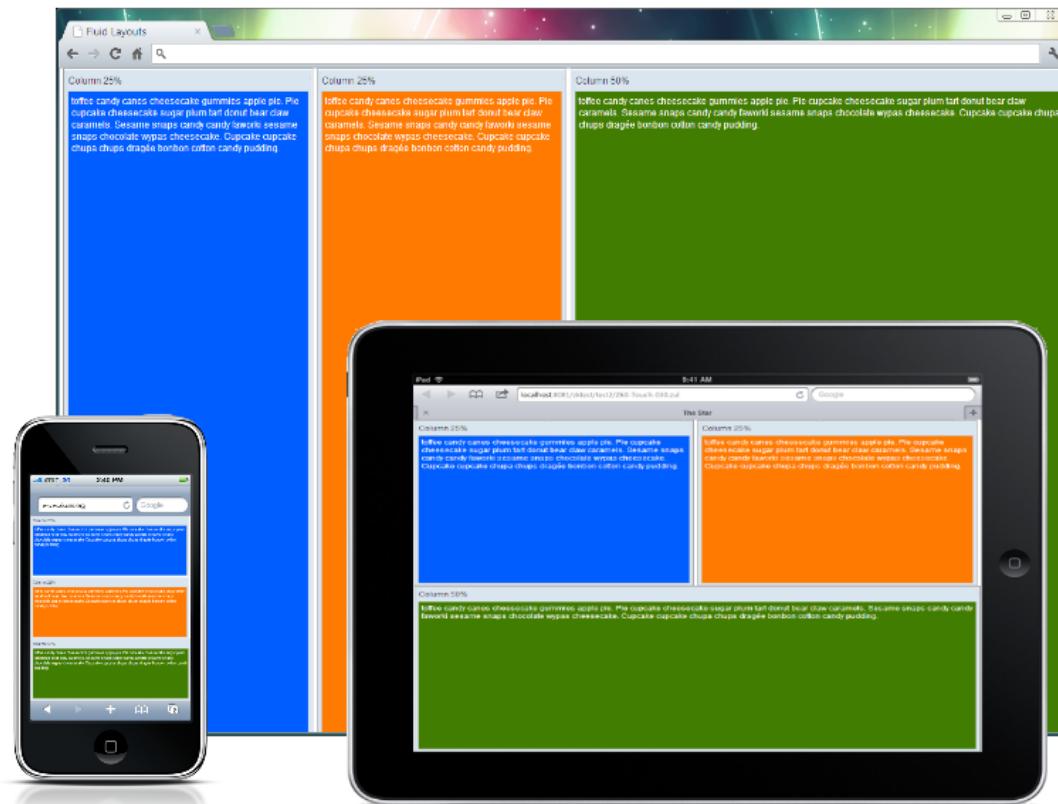
The above example code can be downloaded here - Github^[2].

Adaptive Layouts

[CSS 3 only^[3]]

The adaptive layout is more advantageous than Fluid Layouts, the problem we met in the fluid layout is that its content can only change to the screen's size, but the layout may break if the screen is not big enough. The adaptive layout can solve this by using CSS 3 Media Query^[4].

For example,



```
<?taglib uri="http://www.zkoss.org/dsp/web/theme" prefix="t"?>
<zk>
<style>
.z-hlayout-inner {
    ${| class='wikitable' | width="100%"}
    height: 100%;
}
.z-hlayout-inner {
    width: 25%;
}
.z-hlayout-inner:last-child {
    width: 50%;
}
@media screen and (max-width: 1024px) {
    .z-hlayout-inner {
        width: 50%;
```

```

        height: 50%;
    }
.z-hlayout-inner:last-child {
    width: 100%;
    display: block;
}
}

@media screen and (max-width: 750px) {
    .z-hlayout-inner {
        width: 100%;
        height: 33%;
        display: block;
    }
}

</style>
<hlayout vflex="1">
    <window title="Column 25%" height="100%" sclass="column1" border="normal">
        toffee candy canes cheesecake gummies apple pie. Pie
        cupcake cheesecake sugar plum tart donut
        bear claw caramels. Sesame snaps candy candy faworki sesame
        snaps chocolate wypas cheesecake.
        Cupcake cupcake chupa chups dragée bonbon cotton candy
        pudding.
    </window>
    <window title="Column 25%" height="100%" sclass="column2" border="normal">
        toffee candy canes cheesecake gummies apple pie. Pie
        cupcake cheesecake sugar plum tart donut
        bear claw caramels. Sesame snaps candy candy faworki sesame
        snaps chocolate wypas cheesecake.
        Cupcake cupcake chupa chups dragée bonbon cotton candy
        pudding.
    </window>
    <window title="Column 50%" height="100%" sclass="column3" border="normal">
        toffee candy canes cheesecake gummies apple pie. Pie
        cupcake cheesecake sugar plum tart donut
        bear claw caramels. Sesame snaps candy candy faworki sesame
        snaps chocolate wypas cheesecake.
        Cupcake cupcake chupa chups dragée bonbon cotton candy
        pudding.
    </window>
</hlayout>
</zk>
```

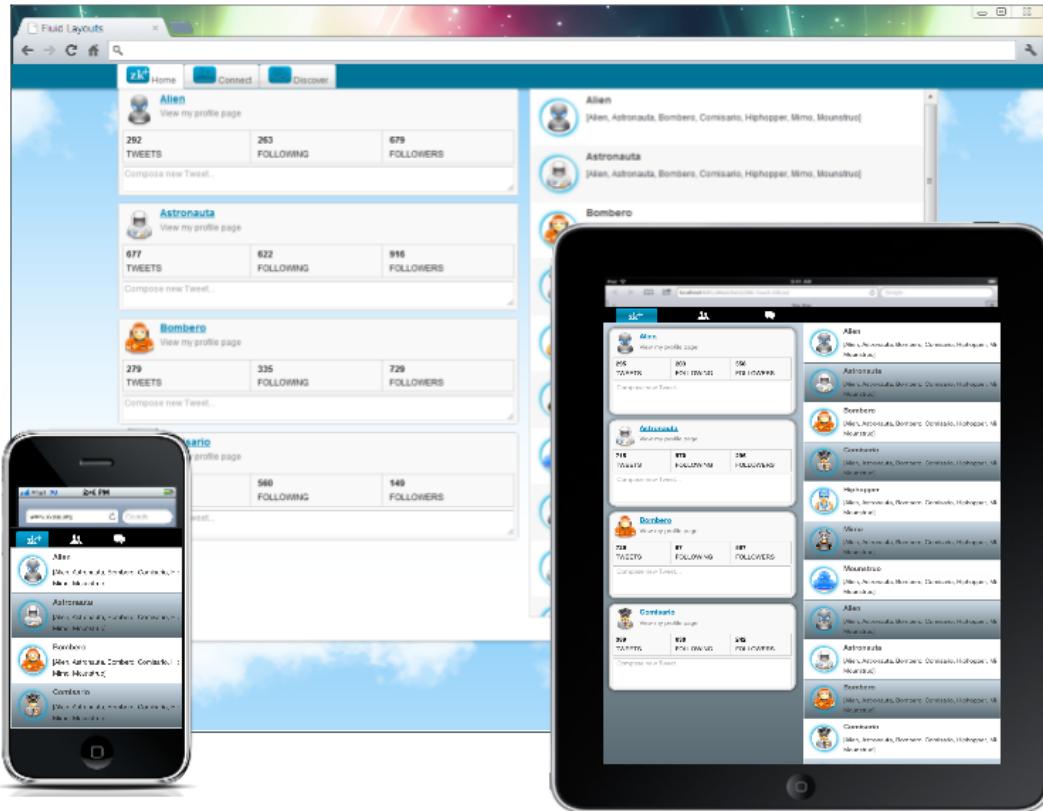
As you can see, we remove the `vflex` and `hflex` for the `Window` component and replace them with a pure CSS style and some condition statements with the `@media` query to switch the layout to fit the screen size. **max-width: 1024px** for ipad or tablet devices and **max-width: 750px** for iphone or smartphones. But those changes are only client effects, what can a ZK developer do in server side if the orientation changes? How many component's stylings need to be scaled when displaying in touch devices? These answers can be found in the following section.

The above example code can be downloaded here - Github^[5].

Responsive Design (mix all)

In ZK 6.5, we refined and polished all components so that they perform seamlessly whether they are on a PC's browser or a Tablet device. In some of the use cases the default styling is not satisfied for user to adjust the layout for different devices and screen sizes, therefore we can employ the ClientInfoEvent to detect whether the browser's orientation changes, and then switch some components' orientation to conform that.

For example,



Zul Source Code

```
<zul>
<zscript><![CDATA[
void doOrientationChange(ClientInfoEvent evt) {
    if ("portrait".equals(evt.getOrientation())) {
        main.setWidth("100%");
        if (evt.getDesktopWidth() < 640)
            sv.setVisible(false);
        Clients.resize(content);
    } else {
        if (!execution.isBrowser("mobile"))
            main.setWidth("80%");
        sv.setVisible(true);
        Clients.resize(content);
    }
}
]]>
```

```
}

]]></zscript>

<tabbox id="main" sclass="main" width="${zk.mobile > 0 ? '100%' : '80%'}"
        vflex="1" onClientInfo="doOrientationChange(event)"
        tabsScroll="false"
        apply="org.zkoss.bind.BindComposer" viewModel="@id('vm')"

@init('TweetsVM')">

    <custom-attributes org.zkoss.zul.image.preload="true" />

    <tabs>
        <tab>
            <caption>
                <div sclass="home" />
                Home
            </caption>
        </tab>
        <tab>
            <caption>
                <image sclass="connect" />
                Connect
            </caption>
        </tab>
        <tab>
            <caption>
                <image sclass="discover" />
                Discover
            </caption>
        </tab>
    </tabs>
    <tabpanel vflex="1" hflex="1">
        <tabpanel vflex="1" hflex="1">
            <hlayout id="content" sclass="main-content" vflex="1">
                <scrollview id="sv" orient="vertical" vflex="1" hflex="1"
                    visible="${zk.mobile > 0}">
                    <children="@init(vm.profiles)">
                        <template name="children" var="profile">
                            <groupbox mold="3d" sclass="profile" hflex="1">
                                <vlayout>
                                    <hlayout>
                                        <image sclass="@bind(profile.ownerIcon)" />
                                    </hlayout>
                                    <vlayout>
                                        <a sclass="fullname" label="@bind(profile.author)" />
                                        <label value="View my profile page" style="color:gray" />
                                    </vlayout>
                                </hlayout>
                                <hlayout sclass="status">
                                    <div sclass="vbar first-vbar" hflex="1">
                                        <label sclass="number" value="@bind(profile.tweets)" />
                                    </div>
                                </hlayout>
                            </groupbox>
                        </template>
                    </children>
                </scrollview>
            </hlayout>
        </tabpanel>
    </tabpanel>

```

```

        <separator />
        <label sclass="text" value="TWEETS" />
    </div>
    <div sclass="vbar" hflex="1">
        <label sclass="number" value="@bind(profile.following)" />
        <separator />
        <label sclass="text" value="FOLLOWING" />
    </div>
    <div sclass="vbar" hflex="1">
        <label sclass="number" value="@bind(profile.followers)" />
        <separator />
        <label sclass="text" value="FOLLOWERS" />
    </div>
</hlayout>
<textbox rows="2" placeholder="Compose new Tweet..." />

```

multiline="true" hflex="1" />

```

</vlayout>
</groupbox>
</template>
</scrollview>
<listbox model="@load(vm.tweets)" vflex="1" hflex="1">
    <template name="model" var="tweet">
        <listitem>
            <listcell>
                <hlayout>
                    <image sclass="@load(tweet.authorIcon)" />
                    <div>
                        <label sclass="author" value="@load(tweet.author)" />
                        <separator />
                        <label sclass="content" multiline="true"
value="@load(tweet.content)" />
                    </div>
                </hlayout>
            </listcell>
        </listitem>
    </template>
</listbox>
</hlayout>
</tabpanel>
</tabpanels>
</tabbox>
</zk>

```

In this example, we layout the page with ZUL Components and only register the *ClientInfoEvent* to handle the display when re-orientating . We manage the main content of the listbox using *vflex* and *hflex* to expand the tweet's

content according to the max height, and then we apply the same concepts mentioned in Adaptive Layouts, with *@Media Query* to fine tune some areas in the page, for example making profile area invisible on smartphones. For an example of this you can refer to the following *CSS Content* section for more details.

Note: Some of the components and features used above are available in ZK EE only [6].

CSS Content

```
<%@ taglib uri="http://www.zkoss.org/dsp/web/core" prefix="c" %>
<%@ taglib uri="http://www.zkoss.org/dsp/web/theme" prefix="t" %>
<%-- For tablet or orientation in portrait devices --%>
@media only screen and (orientation:portrait) {
    body {
        margin: 0;
        padding: 0;
        ${t:gradient('ver', '#cedce7 0%;#596a72 100%')};
    }
    <%-- Customize the default tabbox styling --%>
    .z-tabs-header {
        height: auto;
        background: black;
    }
    .z-tabs-cnt > li.z-tab,
    .z-tabs-cnt > .z-tab:active {
        background: transparent;
        ${t:boxShadow('none')};
        border: 0;
        width: 128px;
        height: 32px;
    }
    .z-tab .z-label {
        display: none;
    }
    .z-tabs-cnt > li.z-tab.z-tab-seld,
    .z-tabs-cnt > li.z-tab.z-tab-seld:first-child,
    .z-tabs-cnt > li.z-tab.z-tab-seld:active,
    .z-tabs-cnt > li.z-tab.z-tab-seld:active:first-child {
        background: black;
        border-color: transparent;
        ${t:boxShadow('1px 1px 0 black')};
    }
    .z-tabs-cnt > .z-tabs {
        background: #555;
    }
    td.z-caption-r {
        text-align: center;
    }
    .main-content {
        max-height: 2048px;
```

```
        }
        <%-- Change the tab styling --%>
        .z-tab .home {
            background: transparent;
        }
        .z-tab-sel'd .home {
            background: ${t:gradValue('ver', '#02ABDE 0%; #007497 50%; #02ABDE 100%')};
        }
        .home:before {
            top: 12px;
            left: 25px;
        }
        .home:after {
            top: -1px;
            left: 30px;
        }
        .connect {
            background:
url(${c:encodeURL('/images/icons/icon_friendrequests_white.png')}) no-repeat center center;
        }
        .discover {
            background:
url(${c:encodeURL('/images/icons/icon_messagestop_white.png')}) no-repeat center center;
        }

        .z-tab .z-image,
        .z-tab .home {
            height: 32px;
            line-height: 28px;
            width: 80px;
        }
        tr.z-listbox-odd {
            ${t:gradient('ver', '#cedce7 0%;#596a72 100%')};
        }
        .z-scrollbar-content-ver:first-child .profile {
            margin: 10px;
        }
        .profile {
            margin-left: 10px;
            border: 3px solid #CFCFCF;
            ${| class='wikitable' | width="100%"}
            ${t:boxShadow('0 0 7px rgba(0, 0, 0, 0.70)' )};
        }
        .z-groupbox-3d-cnt {
```

```
        border: 0px;
    }

}

<%-- For smartphones or small screen --%>
@media screen and (orientation:portrait) and (max-width: 720px) {

    .main-content {
        max-height: 1024px;
    }

    .z-tabs-cnt > li.z-tab,
    .z-tabs-cnt > .z-tab:active {
        background: transparent;
        ${t:boxShadow('none')};
        border: 0;
        width: 80px;
        height: 32px;
    }

    .z-tab .z-image,
    .z-tab .home {
        height: 32px;
        line-height: 28px;
        width: 60px;
    }

    .home:before {
        top: 12px;
        left: 20px;
    }

    .home:after {
        top: -1px;
        left: 25px;
    }

    .main-content > .z-hlayout-inner:first-child {
        display: none;
    }
}
```

The above example code can be downloaded here - [Github](#)^[7].

Related Articles

We also recommend to read the articles below:

- [Responsive_Design_in_ZK_Part_1](#)
- [Responsive_Design_in_ZK_Part_2](#)
- [Responsive_Design_in_ZK_Part_3](#)
- [Use_Bootstrap_with_ZK:_Responsive_Admin_Template](#)

Version History

Version	Date	Content
6.5.0	September, 2012	ZK 6.5.0 Release

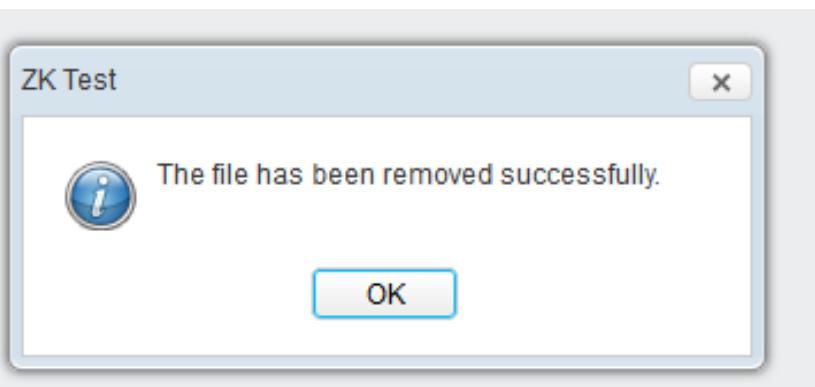
References

- [1] http://en.wikipedia.org/wiki/Responsive_Web_Design
- [2] <https://github.com/jumperchen/ZKResponsiveDesign/blob/master/src/main/webapp/layout/layout1.zul>
- [3] http://www.w3schools.com/cssref/css3_browsersupport.asp
- [4] <http://www.w3.org/TR/css3-mediaqueries/>
- [5] <https://github.com/jumperchen/ZKResponsiveDesign/blob/master/src/main/webapp/layout/layout2.zul>
- [6] <http://www.zkoss.org/product/edition.dsp>
- [7] <https://github.com/jumperchen/ZKResponsiveDesign/blob/master/src/main/webapp/layout/layout3.zul>

Message Box

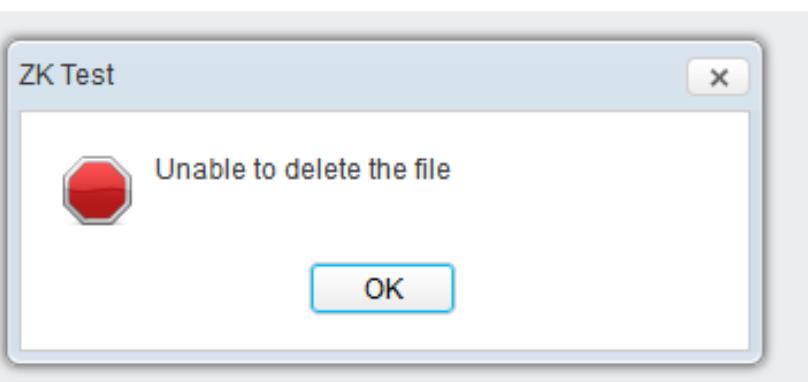
In addition to composing your own window for displaying a message, ZK provides a simple utility: Messagebox [1][2]. For example,

```
Messagebox.show("The file has been removed successfully.");
```



You could specify a different icon for different scenarios, such as alerting an error:

```
Messagebox.show("Unable to delete the file", null, 0, Messagebox.ERROR);
```



Another typical use is to confirm the users for a decision, such as

```
Messagebox.show("Are you sure you want to remove the file?", null,
    Messagebox.YES+Messagebox.NO, Messagebox.QUESTION,
```

```

new EventListener<MouseEvent>() {
    public void onEvent(MouseEvent event) {
        if (Messagebox.ON_YES.equals(event.getName()) )
            //delete the file
    }
} );

```

ZK will close the message box when a user clicks a button, you don't need to call a method to close it.

Notice that the invocation of `show()` returns immediately without waiting for a user's clicking^[3].

There are a lot more options for a message box, such as the button's position and the label. Please refer to ZK Component Reference: Messagebox and Messagebox^[1] for more information.

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

[2] If you are using zscript, there is a shortcut called `alert` as follows UNIQ-source-0-736dd3f9533a2ca2-QINU

[3] If you enable the event thread, the invocation will be stalled until the user clicks a button. It is easier but threading is not cheap. For more information, please refer to the Event Threads section.

Close on Exceptions

When an exception happens inside a message box's event listener and is handled by a ZK error page. The message box doesn't disappear, and it might cover the information on the error page. To avoid this, you can:

- create a modal window in an error page, then the modal window will be on top of the message box.
- catch an exception inside a message box's event listener and detach the message box.

Layouts and Containers

Layouts are components used to partition the display area it owns into several sub-areas for its child components, while containers *group* its child components into the display area it owns.

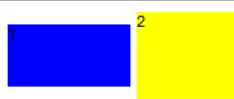
Users are allowed to nest one from another to create desired UI.

Layouts

This section provides brief introductions for some of the layout components in ZK. For detailed information and the complete list of layouts, please refer to ZK Component Reference: Layouts.

Hlayout and Vlayout

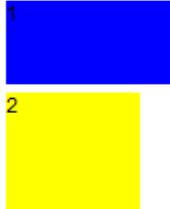
Hlayout and Vlayout are simple and light-weighted layout components that arrange their children to be displayed horizontally and vertically respectively. Also, they are easily customizable as they are made up of HTML DIVs.



```

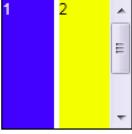
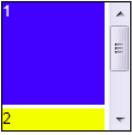
<hlayout>
    <div width="100px" height="50px" style="background:blue">1</div>
    <div width="80px" height="70px" style="background:yellow">2</div>
</hlayout>

```

	<pre><vlayout> <div width="100px" height="50px" style="background:blue">1</div> <div width="80px" height="70px" style="background:yellow">2</div> </vlayout></pre>
---	--

Scrolling

- To make Hlayout and Vlayout scrollable, specify "overflow:auto;" to "style" .
- The height of Hlayout and Vlayout depends on the size of their children, therefore, in order to keep the height of Hlayout and Vlayout constant for the scroll bar to appear, specify a fixed height to Hlayout and Vlayout or place them into a fixed height container, EX: "<window height="100px"..." .

	<pre><hlayout width="100px" height="100px" style="border:1px solid black;overflow:auto;"> <div width="40px" height="150px" style="background:blue;color:white;">1</div> <div width="40px" height="150px" style="background:yellow;">2</div> </hlayout></pre>
	<pre><vlayout width="100px" height="100px" style="border:1px solid black;overflow:auto;"> <div width="80px" height="80px" style="background:blue;color:white;">1</div> <div width="80px" height="80px" style="background:yellow;">2</div> </vlayout></pre>

Alignment

Users are allowed to change sclass to control alignment.

Text: <input type="text"/> win Text: <input type="text"/> win Text: <input type="text"/> win	<pre> <zk> <hlayout sclass="z-valign-top"> <label value="Text:"/> <textbox/> <window width="50px" height="50px" title="win" border="normal"/> </hlayout> <separator/> <hlayout> <label value="Text:"/> <textbox/> <window width="50px" height="50px" title="win" border="normal"/> </hlayout> <separator/> <hlayout sclass="z-valign-bottom"> <label value="Text:"/> <textbox/> <window width="50px" height="50px" title="win" border="normal"/> </hlayout> </zk> </pre>
--	--

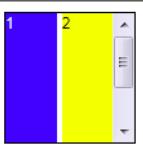
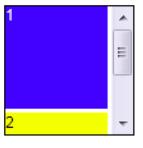
Hbox and Vbox

Similar to Hlayout and Vlayout, Hbox and Vbox arrange their children to be displayed horizontally and vertically respectively. Hbox and Vbox provide more functionalities such as splitter, align and pack. However, their **performance is slower**, so it is suggested to use Hlayout and Vlayout if you'd like to use them a lot in a UI, unless you need the features that only Hbox and Vbox support.

	<pre><hbox> <div width="100px" height="50px" style="background:blue">1</div> <splitter collapse="before"/> <div width="80px" height="70px" style="background:yellow">2</div> </hbox></pre>
	<pre><vbox> <div width="100px" height="50px" style="background:blue">1</div> <splitter collapse="after"/> <div width="80px" height="70px" style="background:yellow">2</div> </vbox></pre>

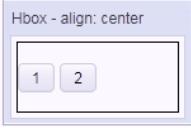
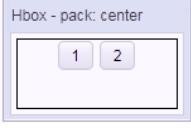
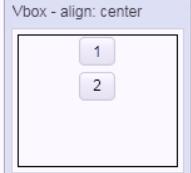
Scrolling

- Hbox and Vbox are created by a table, however, HTML tables are not able to show scroll bars. Hence, to achieve this, users will need to place them in a scrolling container.

	<pre><div width="100px" height="100px" style="border:1px solid black;overflow:auto;"> <hbox> <div width="40px" height="150px" style="background:blue;color:white;">1</div> <div width="40px" height="150px" style="background:yellow;">2</div> </hbox> </div></pre>
	<pre><div width="100px" height="100px" style="border:1px solid black;overflow:auto;"> <vbox> <div width="80px" height="80px" style="background:blue;color:white;">1</div> <div width="80px" height="80px" style="background:yellow;">2</div> </vbox> </div></pre>

Alignment

- Users are also allowed to specify align and pack to control alignment.

	<pre><window title="Hbox" border="normal" width="150px" height="100px"> <caption label="align: center" /> <hbox width="100%" height="100%" style="border:1px solid black;" align="center"> <button label="1" /> <button label="2" /> </hbox> </window></pre>
	<pre><window title="Hbox" border="normal" width="150px" height="100px"> <caption label="pack: center" /> <hbox width="100%" height="100%" style="border:1px solid black;" pack="center"> <button label="1" /> <button label="2" /> </hbox> </window></pre>
	<pre><window title="Vbox" border="normal" width="150px" height="150px"> <caption label="align: center" /> <vbox width="100%" height="100%" style="border:1px solid black;" align="center"> <button label="1" /> <button label="2" /> </vbox> </window></pre>
	<pre><window title="Vbox" border="normal" width="150px" height="150px"> <caption label="pack: center" /> <vbox width="100%" height="100%" style="border:1px solid black;" pack="center"> <button label="1" /> <button label="2" /> </vbox> </window></pre>

For more detailed information, please refer to Hbox and Vbox.

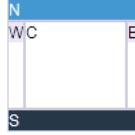
- Users are also allowed to use "cell" to control each cell's alignment.

	<pre><hbox width="500px"> <cell style="border:1px solid black;"> <button label="Help"/> </cell> <cell style="border:1px solid black; hflex=6 align=center"> <button label="Add"/> <button label="Remove"/> <button label="Update"/> </cell> <cell style="border:1px solid black; hflex=4 align=right"> <button label="OK"/> <button label="Cancel"/> </cell> </hbox></pre>
---	--

	<pre> <vbox width="300px" align="stretch"> <cell style="border:1px solid black;"> <button label="Help"/> </cell> <cell style="border:1px solid black;" align="center"> <button label="Add"/> <button label="Remove"/> <button label="Update"/> </cell> <cell style="border:1px solid black;" align="right"> <button label="OK"/> <button label="Cancel"/> </cell> </vbox> </pre>
---	--

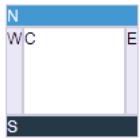
Borderlayout

Borderlayout divides its child components into to five areas: North, South, East, West and Center. The heights of North and South are first decided, the remaining space is then given to Center as its height. Note that East and West also take on the height of Center.

	<pre> <borderlayout width="100px" height="100px"> <north> <div style="background:#008db7;color:white;">N</div> </north> <south> <div style="background:#112f37;color:white;">S</div> </south> <center> <div>C</div> </center> <east> <div style="background:#f2f2f2;">E</div> </east> <west> <div style="background:#f2f2f2;">W</div> </west> </borderlayout> </pre>
--	--

flex

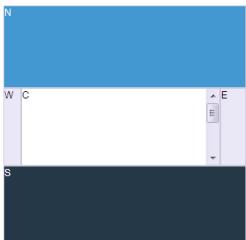
Layout region shares the height of Borderlayout with a distributing sequence of: North, South and Center while the heights of East and West take on the height of Center. In the previous sample, the div in the layout region does not take up all of layout region's space. In order for the child to occupy the whole area, please set vflex="1" to the child component.



```
<borderlayout width="100px" height="100px">
    <north>
        <div style="background:#008db7;color:white;">N</div>
    </north>
    <south>
        <div style="background:#112f37;color:white;">S</div>
    </south>
    <center>
        <div>C</div>
    </center>
    <east>
        <div vflex="1" style="background:#f2f2f2;">E</div>
    </east>
    <west>
        <div vflex="1" style="background:#f2f2f2;">W</div>
    </west>
</borderlayout>
```

Scrolling

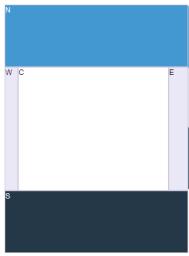
- The height of Center depends on Borderlayout but not on its child, therefore, the height of Center will not be expanded by the growing size of its child components. If Center's height is too short for its child, Center will cut out the contents of its child, hence, to avoid this, specify autoscroll="true" to Center in order to assign Center to handle the scrolling.



```
<font size="7.76">
<borderlayout width="300px" height="300px">
    <north>
        <div height="100px" style="background:#008db7;color:white;">N</div>
    </north>
    <south>
        <div height="100px" style="background:#112f37;color:white;">S</div>
    </south>
    <center autoscroll="true">
        <div height="200px">C</div>
    </center>
    <east flex="true">
        <div width="30px" style="background:#f2f2f2;">E</div>
    </east>
    <west flex="true">
        <div width="20px" style="background:#f2f2f2;">W</div>
    </west>
</borderlayout>
</font>
```

Grown by children

- To make Borderlayout dependent on the size of its child components, vflex feature is applied. Specify vflex="min" to each layout region and Borderlayout.



```

<font size="8.19">
<borderlayout width="300px" vflex="min">
    <north vflex="min">
        <div height="100px" style="background:#008db7;color:white;">N</div>
    </north>
    <south vflex="min">
        <div height="100px" style="background:#112f37;color:white;">S</div>
    </south>
    <center vflex="min">
        <div height="200px">C</div>
    </center>
    <east flex="true">
        <div width="30px" style="background:#f2f2f2;">E</div>
    </east>
    <west flex="true">
        <div width="20px" style="background:#f2f2f2;">W</div>
    </west>
</borderlayout>
</font>

```

Borderlayout in a container

- Almost all containers' heights depend on their child components, however, the height of Borderlayout does not expand according to the sizes of its child components, therefore, when placing Borderlayout in a container, users have to specify a fixed height in order for Borderlayout to be visible.

```

<zK>
    <window title="win" border="normal">
        <borderlayout height="200px">
            <north>
                <div style="background:blue">N</div>
            </north>
            <south>
                <div style="background:blue">S</div>
            </south>
            <center>
                <div>C</div>
            </center>
            <east>
                <div style="background:yellow">E</div>
            </east>
            <west>
                <div style="background:yellow">W</div>
            </west>
        </borderlayout>
    </window>
</zK>

```

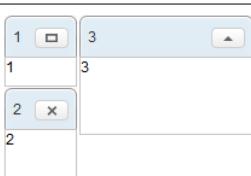
- The default height of BorderLayout is dependent on its parent component, therefore, users can also put BorderLayout in a container with a fixed height.

```
<zk>
    <window title="win" border="normal" height="200px">
        <borderlayout>
            <north>
                <div style="background:blue">N</div>
            </north>
            <south>
                <div style="background:blue">S</div>
            </south>
            <center>
                <div>C</div>
            </center>
            <east>
                <div style="background:yellow">E</div>
            </east>
            <west>
                <div style="background:yellow">W</div>
            </west>
        </borderlayout>
    </window>
</zk>
```

Columnlayout

Columnlayout places its child components into multiple columns while each column allows any number of child components placed vertically with different heights (but with the same widths). Unlike portallayout, Columnlayout does *not allow* end users the ability to move child components to different locations at will (although of course, developers are allowed to use the ZK application to re-arrange the order of children components).

- Available for ZK:
- CE** **PE** **EE**



```
<font size="8.46">
<columnlayout>
    <columnchildren width="30%" style="padding: 5px 1px">
        <panel height="60px" title="1" border="normal" maximizable="true">
            <panelchildren>1</panelchildren>
        </panel>
        <panel height="80px" title="2" border="normal" closable="true">
            <panelchildren>2</panelchildren>
        </panel>
    </columnchildren>
    <columnchildren width="70%" style="padding: 5px 1px">
        <panel height="100px" title="3" border="normal" collapsible="true">
            <panelchildren>3</panelchildren>
        </panel>
    </columnchildren>
</columnlayout>
</font>
```

Portallayout

Portallayout places its child components into multiple columns while each column can allow any number of child components to be placed vertically with different heights (but with the same widths). Users are also allowed to move any of them to any area desired like that of a portal.

- Available for ZK:

- **CE** **PE** **EE**



```
<font size="8.44">
<portallayout>
  <portalchildren width="40%" style="padding: 5px 1px">
    <panel height="60px" title="1" border="normal" maximizable="true">
      <panelchildren>1</panelchildren>
    </panel>
    <panel height="90px" title="2" border="normal" closable="true">
      <panelchildren>2</panelchildren>
    </panel>
  </portalchildren>
  <portalchildren width="60%" style="padding: 5px 1px">
    <panel height="100px" title="3" border="normal" collapsible="true">
      <panelchildren>3</panelchildren>
    </panel>
    <panel height="55px" title="4" border="normal" closable="true">
      <panelchildren>4</panelchildren>
    </panel>
  </portalchildren>
</portallayout>
</font>
```

Tablelayout

Tablelayout places its child components in a table. This implementation is based on an HTML TABLE tag.

- Available for ZK:



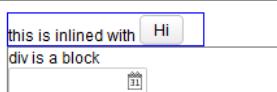
```
<tablelayout columns="2">
  <tablechildren>
    <panel title="1" border="normal"
      collapsible="true" width="80px" height="60px">
      <panelchildren>1</panelchildren>
    </panel>
  </tablechildren>
  <tablechildren>
    <panel title="2" border="normal"
      collapsible="true" width="80px" height="60px">
      <panelchildren>2</panelchildren>
    </panel>
  </tablechildren>
  <tablechildren>
    <panel title="3" border="normal"
      collapsible="true" width="80px" height="60px">
      <panelchildren>3</panelchildren>
    </panel>
  </tablechildren>
  <tablechildren>
    <panel title="4" border="normal"
      collapsible="true" width="80px" height="60px">
      <panelchildren>4</panelchildren>
    </panel>
  </tablechildren>
</tablelayout>
```

Containers

This section provides a brief introduction for some of the container components in ZK. For detailed information and a complete list of containers, please refer to ZK Component Reference: Containers.

Div and Span

Div and span are the most light-weighted containers to group child components. They work the same way as HTML DIV and SPAN tags respectively. Div is a block element that would cause line break for the following sibling i.e. the child and its sibling won't be on the same line (horizontal position). On the other hand, span is an *inline* element which would place the child component and its siblings on the same line (horizontal position).



```
<div style="border: 1px solid blue" width="150px">
  this is
  <span>inlined with <button label="Hi"/></span>
</div>
<div style="border: 1px solid grey">
  <div>div is a block</div>
  <datebox/>
</div>
```

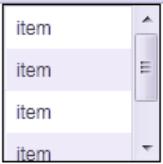
Scrolling

Span:

- Span is an inline element that is not scrollable.

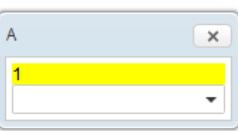
Div:

- To make Div scrollable, specify "overflow:auto;" to "style".
- The height of Div depends on the size of its children, therefore, in order to keep the height of Div constant for the scroll bar to appear, specify a fixed height to Div.

	<pre><div height="100px" width="100px" style="border:1px solid black;overflow:auto;"> <grid> <rows> <row>item</row> <row>item</row> <row>item</row> <row>item</row> <row>item</row> </rows> </grid> </div></pre>
---	--

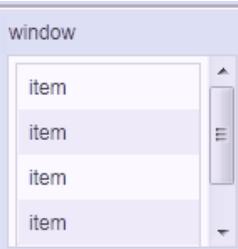
Window

Window is a container providing captioning, bordering, overlapping, draggable, closable, sizable, and many other features. Window is also the owner of an ID space, such that each child component and its IDs are in one independent window so as to avoid the IDs of child components conflicting with one another.

	<pre><window title="A" closable="true" sizable="true" border="normal" mode="overlapped"> <div style="background: yellow">1</div> <combobox/> </window></pre>
---	--

Scrolling

- To make Window scrollable, specify "overflow:auto;" from "contentStyle".
- The height of Window is dependent on the size of its children, therefore, in order to keep the height of Window constant for the scroll bar to appear, specify a fixed height to Window.

	<pre><window title="window" border="normal" height="150px" width="150px" contentStyle="overflow:auto;"> <grid> <rows> <row>item</row> <row>item</row> <row>item</row> <row>item</row> <row>item</row> </rows> </grid> </window></pre>
---	---

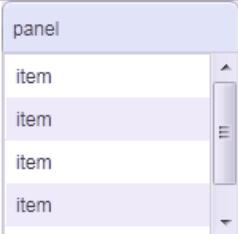
Panel

Like Window, panel is another powerful container supporting captioning, bordering, overlapping and many other features. However, IdSpace^[1] is not implemented by this component, therefore, all of its children belong to the same ID space of its parent.

	<pre><panel title="A" framable="true" border="normal" maximizable="true" collapsible="true"> <panelchildren> <div style="background: yellow">1</div> <combobox/> </panelchildren> </panel></pre>
---	--

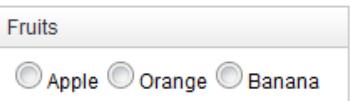
Scrolling

- To make Panel scrollable, specify "overflow:auto;" to "style" of "panelchildren".
- The height of Panel is dependent on the size of its children, therefore, in order to keep the height of the Panel constant for the scroll bar to appear, specify a fixed height to Panel.

	<pre><panel title="panel" border="normal" height="150px" width="150px"> <panelchildren style="overflow:auto;"> <grid> <rows> <row>item</row> <row>item</row> <row>item</row> <row>item</row> <row>item</row> </rows> </grid> </panelchildren> </panel></pre>
--	--

Groupbox

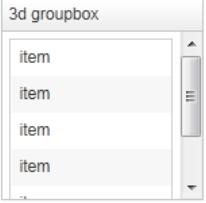
Groupbox is a light-weighted way to group child components together. It supports "caption" and "border", however, it does not support overlapping or resizing. Like Panel, IdSpace^[1] is not implemented by this component either.

	<pre><groupbox mold="3d"> <caption label="Fruits"/> <radiogroup> <radio label="Apple"/> <radio label="Orange"/> <radio label="Banana"/> </radiogroup> </groupbox></pre>
---	---

Scrolling

3d mold only

- To make Groupbox scrollable, specify "overflow:auto" to "contentStyle".
- The height of the Groupbox depends on the size of its children, therefore, in order to keep the height of the Groupbox constant for the scroll bar to appear, specify a fixed height to Groupbox.

	<pre><groupbox mold="3d" height="150px" width="150px" contentStyle="overflow:auto;" > <caption label="3d groupbox" /> <grid> <rows> <row forEach="1, 2, 3, 4, 5, 6">item</row> </rows> </grid> </groupbox></pre>
---	--

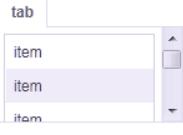
Tabbox

Tabbox is a container used to display a set of tabbed groups of components. A row of tabs can be displayed at the top (or left) of the tabbox; users can switch between each tab group by a simple click. IdSpace [1] is not implemented by this component either.

	<pre><tabbox height="80px"> <tabs> <tab label="Tab 1"/> <tab label="Tab 2"/> </tabs> <tabpanels> <tabpanel>This is panel 1</tabpanel> <tabpanel>This is panel 2</tabpanel> </tabpanels> </tabbox></pre>
--	---

Scrolling

- To make Tabpanel scrollable, specify "overflow:auto;" to "style".
- The height of Tabpanel is dependent on the size of its children, therefore, in order to keep the height of the Tabpanel constant for the scroll bar to appear, specify a fixed height to Tabbox.

	<pre><tabbox height="100px" width="150px"> <tabs> <tab label="tab" /> </tabs> <tabpanels> <tabpanel style="overflow:auto;" > <grid> <rows> <row forEach="1, 2, 3, 4, 5, 6">item</row> </rows> </grid> </tabpanel> </tabpanels> </tabbox></pre>
---	--

Hflex and Vflex

Hflex (`HtmlBasedComponent.setHflex(java.lang.String)`^[1]) and vflex (`HtmlBasedComponent.setVflex(java.lang.String)`^[2]) indicate the flexibility of the component, which indicates how a component's parent distributes the remaining empty space among its children. Hflex controls the flexibility in the horizontal direction, while vflex in the vertical direction.

Flexible components grow and shrink to fit their given space. Components with larger flex values will be made larger than components with lower flex values, at the ratio determined by the two components. The actual value is not relevant unless there are other flexible components within the same container. Once the default sizes of components in a box are calculated, the remaining space in the box is divided among the flexible components, according to their flex ratios. Specifying a flex value of 0 has the same effect as leaving the flex attribute out entirely.

2 Different Underlying Implementations

ZK 9 implements hflex/vflex in a whole new, more performant way -- by CSS3 flexbox^[3], which is supported by modern browsers natively. With this change, it doesn't calculate an element's size in javascript thus improving the client-side performance. This change should be transparent for developers.

The exceptional case is `min`, e.g. `hflex="min"` or `vflex="min"`, which still sets width by JavaScript.

Fall Back to the Old Way

However, if your application depends on the previous implementation, you can fall back by the property `org.zkoss.zul.css.flex="false"`.

Prerequisite: Parent Requires Width/Height Specified

Notice that, if the parent has no predefined size (width/height) (i.e., its size is decided by its children), the flexible component won't take any space. For example, the inner div (with vflex) in the following example takes no space:

```
<div><!--Wrong! The height is required since it is minimal height by default-->
  <datebox width="150px"/>
  <div vflex="1" style="background: yellow"/><!--height will be zero since height not specified in parent div-->
</div>
```

To solve it, you have to specify the height in the outer div, such as `<div height="100%">`, `<div height="200px">`, or `<div vflex="1">`.

Fit-the-Rest Flexibility

The simplest use of flex is to have one component to take the rest of the space of its parent (or the page, if it is the root component). For example,

```
<zk>
  <datebox/>
  <div vflex="1" style="background: yellow"/>
</zk>
```

And, the result



Here is another example that we'd like to grow the tabbox to fit the rest of the space:

```
<zk>
    <datebox/>
    <tabbox vflex="1">
        <tabs>
            <tab label="Home"/>
            <tab label="Direction"/>
        </tabs>
        <tabpanels>
            <tabpanel style="overflow: auto">
                <div height="500px" width="100%" style="background: yellow"/>
            </tabpanel>
            <tabpanel>
            </tabpanel>
        </tabpanels>
    </tabbox>
</zk>
```

Notice you could specify `style="overflow: auto"` in thetabpanel such that the scrollbar will be inside the tabbox rather than the browser window if the content is too large to fit.



Proportional Flexibility

The absolute value of the vflex/hflex is not that important. It is used to determine the proportion among flexible components. That is, you can give different integers to differentiate child components so they will take space proportionally per the given vflex/hflex value. For example,

```
<div width="200px" height="50px">
    <div style="background: blue" vflex="1" hflex="1"/>
    <div style="background: yellow" vflex="2" hflex="1"/>
</div>
```

And, the result is



Here is another example (hflex):

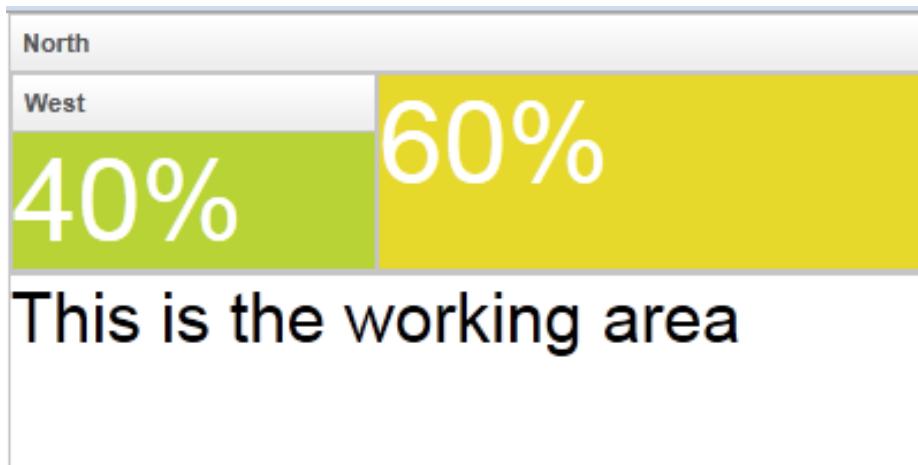
```
<hlayout width="200px">
    <div style="background: blue" hflex="1">1</div>
    <div style="background: yellow" hflex="2">2</div>
</hlayout>
```



Minimum Flexibility

Sometimes, you might wish that the parent component's size is determined by its children. Or I shall say, the size of the parent component is just high/wide enough to hold all of its child components. Specifying vflex/hflex="min" can fulfill this fit-the-content requirement.

```
<borderlayout height="200px" width="400px">
    <north title="North" vflex="min">
        <borderlayout vflex="min">
            <west title="West" size="40%" flex="true" vflex="min">
                <div style="background:#B8D335">
                    <label value="40%" style="color:white;font-size:50px"/>
                </div>
            </west>
            <center flex="true" vflex="min">
                <div style="background:#E6D92C">
                    <label value="60%" style="color:white;font-size:50px"/>
                </div>
            </center>
        </borderlayout>
    </north>
    <center>
        <label value="This is the working area"
              style="font-size:30px" />
    </center>
</borderlayout>
```



As you can see, the height of the north region of the outer borderlayout is determined by its child borderlayout. And the height of the inner borderlayout, in this example, is determined by the height of its west child region.

Also notice that the flex property (LayoutRegion.setFlex(boolean) [4]) is unique to borderlayout (north and others). Don't confuse it with hflex or vflex.

Don't specify Minimum on a parent and 1 on a child

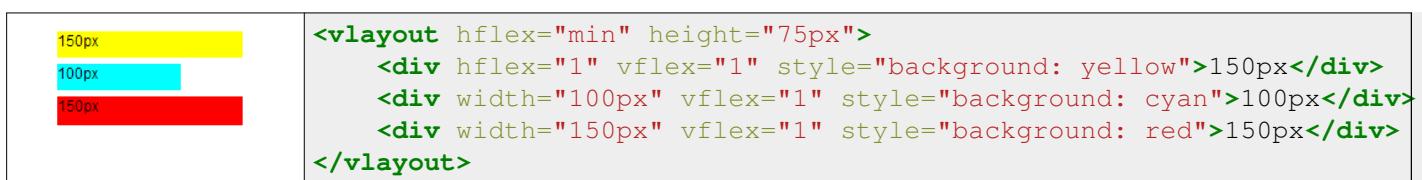
Because min means "calculate the size by its children" and 1 means "calculate the size by its parent", this configuration will make 2 components' **size calculation depends on each other** and get 0 finally. But there are workarounds, please read the following sections.

Component width within Vlayout/Vbox using minimum hflex

In the case below, we see nothing for the incorrect usage (min on the parent - vlayout, 1 on the child - div):

```
<vlayout hflex="min" height="30px">
    <div hflex="1" vflex="1" style="background: yellow"></div>
</vlayout>
```

However, in the case below, because one of the children, red div has a fixed width, vlayout can determine its width. So that yellow div can also determine its width upon its parent, which is 150px.



Component height within Hlayout/Hbox using minimum vflex

Normally, if the siblings of yellow **div** have been defined height correctly, that yellow **div** height should be equal to the **max height** of siblings, which is 30px in the following sample.

<pre><hlayout width="100px" vflex="min"> <div hflex="1" vflex="1" style="background: yellow">30px</div> <div hflex="1" height="20px" style="background: cyan">20px</div> <div hflex="1" height="30px" style="background: red">30px</div> </hlayout></pre>		

However, in the following use case, we should see nothing as it is an incorrect usage:

```
<hlayout width="100px" vflex="min">
    <div hflex="1" vflex="1" style="background: yellow"></div>
</hlayout>
```

Grid's Column and Flexibility

If hflex is specified in the header of grid, listbox and tree, it is applied to the whole column (including the header and contents).

For example, we could assign 33% to the first column and 66% to the second as follows.

```
<grid width="300px">
    <columns>
        <column label="Name" hflex="1"/>
        <column label="Value" hflex="2"/>
    </columns>
    <rows>
        <row>username:<textbox hflex="1"/></row>
        <row>password:<textbox hflex="1"/></row>
    </rows>
</grid>
```

The result is

Name	Value
username:	<input type="text"/>
password:	<input type="text"/>

Notice that we also specify `hflex="1"` to the textbox, so it will take up the whole space.

Alignment

When we create a form, we will put some input elements in a Grid. We can set hflex="min" to Grid and each Column to keep Grid with minimal size.

Name:	<input type="text"/>
Birthday:	<input type="date"/> 31

```
<grid hflex="min">
  <columns>
    <column hflex="min" align="right"/>
    <column hflex="min"/>
  </columns>
  <rows>
    <row>
      <label value="Name:"/>
      <textbox/>
    </row>
    <row>
      <label value="Birthday:"/>
      <datebox/>
    </row>
  </rows>
</grid>
```

If we need the Datebox's width the same as Textbox, we can specify hflex="1" to Datebox.

Name:	<input type="text"/>
Birthday:	<input type="date"/> 31

```
<grid hflex="min">
  <columns>
    <column hflex="min" align="right"/>
    <column hflex="min"/>
  </columns>
  <rows>
    <row>
      <label value="Name:"/>
      <textbox/>
    </row>
    <row>
      <label value="Birthday:"/>
      <datebox hflex="1"/>
    </row>
  </rows>
</grid>
```

Cell colspan

Sometimes we need to put some elements in cross column, we can put it in a Cell and set hflex="1" to the element.

	<pre> <grid hflex="min"> <columns> <column hflex="min" align="right" /> <column hflex="min" /> <column hflex="min" align="right" /> <column hflex="min" /> </columns> <rows> <row> <label value="Name:" /> <textbox/> <label value="Birthday:" /> <datebox/> </row> <row> <label value="Address:" /> <cell colspan="3"> <textbox rows="5" hflex="1"/> </cell> </row> </rows> </grid> </pre>
--	---

For a complete list of controls that you could apply to the columns of grid, listbox and tree, please refer to ZK Developer's Reference/UI Patterns/Grid's Columns and Hflex.

Flexibility versus Percentage

The use of hflex and vflex is similar to the use of percentage in width and height. For example,

```

<div width="200px" height="200px">
    <div height="33%" style="background: blue">1</div>
    <div height="66%" style="background: yellow">2</div>
</div>

```

The advantage of percentage is that the performance will be a little better, since it is done by the browser. However, hflex and vflex are recommended because of the following issues:

- The use of 100% will cause overflow (and then scrollbar appears if overflow:auto), if padding is not zero. Moreover, some browsers might show mysterious scrollbars or overflow the parent's space even if padding is zero.
- The percentage does *not* work, if any of the parent DOM element does not specify the width or height.
- The percentage does *not* support *take-the-rest-space*. For example, the following doesn't work:

```

<!-- a vertical scrollbar appear (not as expected) -->
<div height="100%">
    <datebox/>
    <div height="100%"/>
</div>

```

Body Height and Padding

By default, ZK's theme configures the document's BODY tag as follows.

```
body {
    height: 100%;
    padding: 0 5px;
}
```

Sometimes you might prefer to add some padding vertically, but it *cannot* be done by changing BODY's styling as follows.

```
body {
    height: 100%;
    padding: 5px; /* WRONG! It causes vertical scrollbar to appear
since the 100% height is used with vertical padding */
}
```

As described in the previous section, a vertical scrollbar will appear, since both the vertical padding and the 100% height are specified.

Solution: you shall *not* change the default CSS styling of BODY. Rather, you could enclose the content with the div component, and then specify vflex="1" and the padding to the div component. For example,

```
<div style="padding: 5px 0" vflex="1">
    <grid>
        <rows>
            <row>aaa</row>
        </rows>
    </grid>
</div>
```

Flexibility and Resizing

Vflex and hflex support resizing. If the parent component or the browser window changes its size to increase or decrease the extra space, the child components with vflex/hflex will recalculate themselves to accommodate the new size.

```
<zkb>
    <zscript><![CDATA[
        int[] str = new int[100];
        for(int i=0;i<100;i++) {
            str[i]=i;
        }
    ]]></zscript>

    <div height="100%" width="300px">
        Top of the Tree
        <tree vflex="1">
            <treetr>
                <treetritem forEach="${str}" label="item${each}" />
            </treetritem>
        </treetr>
    </div>
</zkb>
```

```

</tree>
<tree vflex="2">
    <treetrue>
        <treetrue>
            <treetrue>
                <treetrue>
                    <treetrue>
                        <treetrue>
                            <treetrue>
                                <treetrue>
                                    <treetrue>
                                        <treetrue>
                                            <treetrue>
                                                <treetrue>
                                                    <treetrue>
                                                        <treetrue>
                                                            <treetrue>
                                                                <treetrue>
                                                                    <treetrue>
                                                                        <treetrue>
                                                                            <treetrue>
                                                                                <treetrue>
                                                                                    <treetrue>
                                                                                        <treetrue>
                                                                                            <treetrue>
                                                                                                <treetrue>
                                                                                                 <treetrue>
                                                                                                     <treetrue>
                                                                                                         <treetrue>
                                                                                                             <treetrue>
                                                                                                                 <treetrue>
                                                                                                                     <treetrue>
                                                                                                                         <treetrue>
                                                                                                                             <treetrue>
                                                                                                                               <treetrue>
                                                                                                                               <treetrue>
                                                                                                                               <treetrue>
                                                                                                                               <treetrue>
                                                                                                                               <treetrue>
                                                                                                                               <treetrue>
................................................................
Bottom of the Tree
</div>
</zk>
```

Note that the height proportion between the two trees is always 1 : 2, when we change the browser height.

Limitations

Span Ignores Width and Height

Span ignores the width and height, so hflex and vflex have no effect on them (unless you specify display:block^[5] -- but it makes it div eventually).

```

<!-- this example does not work -->
<div width="200px">
    <span style="background: blue" hflex="1">1</span>
    <span style="background: yellow" hflex="2">2</span>
</div>
```

And, the result is as follows - the width has no effect:

1 2

This limitation can be solved by the use of hlayout and div as follows.

```

<!-- this is correct -->
<hlayout width="200px">
    <div style="background: blue" hflex="1">1</div>
    <div style="background: yellow" hflex="2">2</div>
</hlayout>
```

1 2

Hflex Must Align Correctly

Hflex will be wrong if a component is not aligned in the same row with its siblings. For example,

```

<div width="200px">
    <div style="background: blue" hflex="1">1</div><!-- not work since it won't be aligned with sibling div -->
    <div style="background: yellow" hflex="2">2</div>
</div>
```

As shown below, the second div is not aligned vertically with the first div, so is the width not as expected:

1 2

This limitation can be solved by use of hlayout and div as shown in the previous subsection.

Input elements have incorrect margin values in WebKit browsers

In WebKit browsers (Chrome, Safari), the left and right margin values of an input element are considered 2px by browsers, where they are really 0px on screen. This may cause hflex to wrongly handle InputElements like textbox, intbox, etc. For example, in the following case the Textbox does not occupy the entire Div width in Chrome:

```
<div width="300px" style="border: 1px solid green">
    <textbox hflex="1" />
</div>
```

You can work around this by specifying Textbox margin to be 0:

```
<style>
    input.nomargin {
        margin-left: 0;
        margin-right: 0;
    }
</style>
<div width="300px" style="border: 1px solid green">
    <textbox sclass="nomargin" hflex="1" />
</div>
```

Minimum Flexibility Doesn't Change a Component's Size Dynamically

When specifying min at hflex/vflex, ZK only sets a component's size once at the page creation. The component doesn't change its size accordingly even if you add or remove its child components (change its content size). Therefore, if you want to resize the component upon its content again, please call Clients.resize(org.zkoss.zk.ui.Component)^[6].

The same rule applies when you change the content of a parent component to minimum hflex/vflex, the parent component doesn't resize itself upon its content. You can need to call Clients.resize(org.zkoss.zk.ui.Component)^[6].

For example,

```
<zk>
    <div id="div" vflex="1" hflex="1" style="background: blue">blue</div>
    <button label="vflex to min">
        <attribute name="onClick"><! [CDATA[
            div.setVflex("min");
            Clients.resize(div);
        ]]></attribute>
    </button>
</zk>
```

References

- [1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlBasedComponent.html#setHflex\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlBasedComponent.html#setHflex(java.lang.String))
- [2] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlBasedComponent.html#setVflex\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlBasedComponent.html#setVflex(java.lang.String))
- [3] https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox
- [4] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/LayoutRegion.html#setFlex\(boolean\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/LayoutRegion.html#setFlex(boolean))
- [5] <http://www.quirksmode.org/css/display.html>
- [6] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#resize\(org.zkoss.zk.ui.Component\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#resize(org.zkoss.zk.ui.Component))

Grid's Columns and Hflex

This section introduces the usage of ZK auto sizing APIs. All of these can apply to the following components:

• Listbox	• Grid	• Tree
• Listhead	• Columns	• Treecols
• Listheader	• Column	• Treecol

Hflex and Width

There are basically two approaches to control the width of a column: width and hflex. They could be specified in the column's header, such as column and listheader.).

While width (HtmlBasedComponent.setWidth(java.lang.String)^[1]) specifies the width in precise number (such as width="100px"), hflex (HtmlBasedComponent.setHflex(java.lang.String)^[1]) specifies the proportional width (such as hflex="1") or the minimal width (such as hflex="min").

Span

If the total width of all columns is smaller than the grid, there will be some whitespace shown at the right side of the grid.

If you prefer to make each column's width a bit wider to cover the whole grid, you could specify span="true" in grid/listbox/tree (such as Grid.setSpan(java.lang.String)^[2]). If you want to make a particular column larger to cover the whole grid, you could specify a number, such as span="2" to expand the third column.

sizedByContent

If you want to make each column as minimal as possible, you could specify hflex="min" for each column. Alternatively, you could specify sizedByContent="true" in the grid/listbox/tree (Grid.setSizedByContent(boolean)^[3]). In other words, it implies the width of a column that is *not* assigned with width and hflex shall be minimal.

In general, you will specify span="true" too to cover the whole grid/listbox/tree.

Use Cases

Here we take listbox with listheaders as an example to show some different use cases.

Data Length : Short

```
<zk>
<zscript><! [CDATA[
String[] msgs2 = {
    "Application Developer's Perspective",
    "Server+client Fusion architecture",
    "Component Developer's Perspective",
    "Execution Flow of Loading a Page",
    "Execution Flow of Serving an Ajax Request",
    "When to Send an Ajax Request"
};
]]></zscript>
<listbox width="800px">
    <listhead>
        <listheader label="Product" />
        <listheader label="Description" />
        <listheader label="Comment" />
    </listhead>
    <listitem>
        <listcell><label value="${msgs2[0]}"></label></listcell>
        <listcell><label value="${msgs2[1]}"></label></listcell>
        <listcell><label value="${msgs2[2]}"></label></listcell>
    </listitem>
    <listitem>
        <listcell><label value="${msgs2[3]}"></label></listcell>
        <listcell><label value="${msgs2[4]}"></label></listcell>
        <listcell><label value="${msgs2[5]}"></label></listcell>
    </listitem>
</listbox>
</zk>
```

- Default** : Data component will show the data correctly, with no width specification. (the exact widths of columns are rendered by browser automatically)

Product	Description	Comment
Application Developer's Perspective	Server+client Fusion architecture	Component Developer's Perspective
Execution Flow of Loading a Page	Execution Flow of Serving an Ajax Request	When to Send an Ajax Request

Proportional Width

You can use the **hflex** we have mentioned in previous section.

```
<listhead>
    <listheader label="Product" hflex="1"/>
    <listheader label="Description" hflex="2"/>
    <listheader label="Comment" hflex="1" />
</listhead>
```

Product	Description	Comment
Application Developer's Perspective	Server+client Fusion architecture	Component Developer's Perspective
Execution Flow of Loading a Page	Execution Flow of Serving an Ajax Request	When to Send an Ajax Request

Minimum Flexibility

In the case of **hflex=min**, column's width will just fit the contents. As you can see, there might be blank space on the right of the listbox.

```
<zscript><! [CDATA[
String[] msgs2 = {
    "Application Developer's Perspective",
    "Very Short Text",
    "Server+client Fusion architecture",
    "Execution Flow of Serving an Ajax Request",
    "Very Short Text",
    "When to Send an Ajax Request (Addition Text )"
};

]]></zscript>
<listbox width="800px">
    <listhead>
        <listheader label="Product" hflex="min"/>
        <listheader label="Description" hflex="min"/>
        <listheader label="Comment" hflex="min" />
    </listhead>
```

Product	Description	Comment	
Application Developer's Perspective	Very Short Text	Server+client Fusion architecture	(Blank Here)
Execution Flow of Serving an Ajax Request	Very Short Text	When to Send an Ajax Request (Addition Text)	

- If you want your contents to fill the whole grid to eliminate the blank space, you can set **span=true** to make it proportionally expanded.

```
<listbox width="800px" span="true">
    <listhead>
        <listheader label="Product" hflex="min"/>
        <listheader label="Description" hflex="min"/>
        <listheader label="Comment" hflex="min" />
    </listhead>
```

Product	Description	Comment
Application Developer's Perspective	Very Short Text	Server+client Fusion architecture
Execution Flow of Serving an Ajax Request	Very Short Text	When to Send an Ajax Request (Addition Text)

- If you want the rest of the space to be assigned to one of the columns, set **span** to a number. The number is 0-based index of columns.

```
<listbox width="800px" span="0">
  <listhead>
    <listheader label="Product" hflex="min" />
    <listheader label="Description" hflex="min" />
    <listheader label="Comment" hflex="min" />
  </listhead>
```

Product	Description	Comment
Application Developer's Perspective	Very Short Text	Server+client Fusion architecture
Execution Flow of Serving an Ajax Request	Very Short Text	When to Send an Ajax Request (Addition Text)

- If you want the size of the Listbox determined by its content, assign **hflex=min** on the Grid, and make sure all the Listheaders either have **hflex=min** or a fixed **width**.

```
<listbox hflex='min'>
  <listhead>
    <listheader label="Product" hflex="min" />
    <listheader label="Description" hflex="min" />
    <listheader label="Comment" hflex="min" />
  </listhead>
```

Product	Description	Comment
Application Developer's Perspective	Very Short Text	Server+client Fusion architecture
Execution Flow of Serving an Ajax Request	Very Short Text	When to Send an Ajax Request (Addition Text)

Data Length : Long

```
<zscript><![CDATA[
String[] msgs2 = {
  "Application Developer's Perspective",
  "Very Long Long Long Long Long Long Long Long
Long Text",
  "Server+client Fusion architecture",
  "Execution Flow of Serving an Ajax Request",
  "Very Long Long Long Long Long Long Long Long
Long Text",
  "When to Send an Ajax Request"
};]]></zscript>
```

- Default** : ZK data component will wrap the text to fit the width of column.

Product	Description	Comment
Application Developer's Perspective	Very Long Text	Server+client Fusion architecture
Execution Flow of Serving an Ajax Request	Very Normal Normal Text	When to Send an Ajax Request (Addition Text)

Scrollbar

When the sum of content width is larger than Grid width. The scroll appears **if and only if**

1. The Columns and Column component are presented.
2. Each of the Column components is given an **hflex** or **width** value.

Specify Width

This is a simple way to avoid text wrapped by given proper width. However, it can be difficult if you don't know the content length beforehand.

```
<listhead>
    <listheader label="Product" width="250px"/>
    <listheader label="Description" width="470px"/>
    <listheader label="Comment" width="280px" />
</listhead>
```

Product	Description	Comment
Application Developer's Perspective	Very Long Text	Server+client Fusion architecture
Execution Flow of Serving an Ajax Request	Very Normal Normal Text	When to Send an Ajax Request (Addition Text)

Minimum Flexibility

- Set **hflex=min** and ZK will calculate the length of content and set proper width to the column accordingly.
- **Notes:** Remember to set every column with **hflex=min** or specify a specific **width**; otherwise those columns without setting minimum hflex or specifying a width could disappear if not enough space in the listbox.

```
<listhead>
    <listheader label="Product" hflex="min" />
    <listheader label="Description" hflex="min" />
    <listheader label="Comment" hflex="min" />
</listhead>
```

Product	Description	Comment
Application Developer's Perspective	Very Long Text	Server+client Fusion architecture
Execution Flow of Serving an Ajax Request	Very Normal Normal Text	When to Send an Ajax Request (Addition Text)

Mixed Flexibility and width

Width + Hflex proportion

```
<listhead>
    <listheader label="Product" width="120px" />
    <listheader label="Description" hflex="2" />
    <listheader label="Comment" hflex="1" />
</listhead>
```

Product	Description	Comment
Application Developer's Perspective	Very Long Text	Server+client Fusion architecture
Execution Flow of Serving an Ajax Request	Very Normal Normal Text	When to Send an Ajax Request (Addition Text)

Width + Hflex min

```
<listhead>
    <listheader label="Product" width="150px" />
    <listheader label="Description" hflex="min" />
    <listheader label="Comment" hflex="min" />
</listhead>
```

Product	Description	Comment
Application Developer's Perspective	Very Long Text	Server+client Fusion architecture
Execution Flow of Serving an Ajax Request	Very Normal Normal Text	When to Send an Ajax Request (A)

Width + Hflex min + Hflex proportion

```
<listhead>
    <listheader label="Product" width="120px" />
    <listheader label="Description" hflex="min" />
    <listheader label="Comment" hflex="1" />
</listhead>
```

Product	Description	Comment
Application Developer's Perspective	Very Long Long Long Long Long Long Long Long Text	Server+client Fusion architecture
Execution Flow of Serving an Ajax Request	Very Long Long Long Long Long Long Long Long Text	When to Send an Ajax Request

Data Length : Dynamic

If users can't control the data size, that means, you should take both the short data and the long data situation into consideration.

Now we have several attributes, rules, and situations, please choose the right solution for your case.

- **(Blank)** : Doesn't matter
- **V** : Must set
- **X** : Must not set
- **!** : Acceptable but something needs notice.

Specification	Span=true	Hflex proportion	Hflex=min	Specific Width
Fill whole Grid		One of these attributes was set	X	!
No Scrollbar Grid		V	X	!
No Content Wrapping Column			V	!
Fill whole Grid + No Scrollbar Grid		V	X	!
Fill whole Grid + No Content Wrapping	V	All Column components should have one of these attributes		

- **!** : Specific width must not be more than grid's width.

Version History

Version	Date	Content
5.0.6	February 2011	New specification of hflex and span.

References

- [1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlBasedComponent.html#setWidth\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlBasedComponent.html#setWidth(java.lang.String))
- [2] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Grid.html#setSpan\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Grid.html#setSpan(java.lang.String))
- [3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Grid.html#setSizedByContent\(boolean\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Grid.html#setSizedByContent(boolean))

Toolips, Context Menus and Popups

The support of toolips, context menus and popups is generic. Any components inherited from XulElement^[1] can handle them in the same way.

To enable any of them, you have to prepare a component representing the customized look, and then specify this component or its ID in the corresponding properties (such as XulElement.setTooltip(java.lang.String)^[2]) in the target component. Then, when the user triggers it (such as moving the mouse over the target component), the component representing the customized look is shown.

The component representing the customized look must be a Popup component or one of derives, such as Menupopup, while the target component (which causes the customized look to show up) can be any kind of component.

What	Condition	API
Toolips ^[3]	When the user moves the mouse point over the target component for a while	XulElement.setTooltip(java.lang.String) ^[2] or XulElement.setTooltip(org.zkoss.zul.Popup) ^[4]
Context Menus	When the user clicks the <i>right</i> button on the target component	XulElement.setContext(java.lang.String) ^[5] or XulElement.setContext(org.zkoss.zul.Popup) ^[6]
Popups	When the user clicks the <i>left</i> button on the target component	XulElement.setPopup(java.lang.String) ^[7] or XulElement.setPopup(org.zkoss.zul.Popup) ^[8]

[1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/impl/XulElement.html#>

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

[3] Notice that if you'd like to have different text for the tooltip (rather than a fully customized look), you shall use XulElement (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/impl/XulElement.html#>) instead (which is easier to use).

[4] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/impl/XulElement.html#setTooltip\(org.zkoss.zul.Popup\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/impl/XulElement.html#setTooltip(org.zkoss.zul.Popup))

[5] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/impl/XulElement.html#setContext\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/impl/XulElement.html#setContext(java.lang.String))

[6] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/impl/XulElement.html#setContext\(org.zkoss.zul.Popup\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/impl/XulElement.html#setContext(org.zkoss.zul.Popup))

[7] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/impl/XulElement.html#setPopup\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/impl/XulElement.html#setPopup(java.lang.String))

[8] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/impl/XulElement.html#setPopup\(org.zkoss.zul.Popup\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/impl/XulElement.html#setPopup(org.zkoss.zul.Popup))

Toolips

To provide a custom tooltip, you could specify the ID of the custom tooltip in the target component's `tooltip` (`XulElement.setTooltip(java.lang.String)` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/impl/XulElement.html#setTooltip\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/impl/XulElement.html#setTooltip(java.lang.String))) or `XulElement.setTooltip(org.zkoss.zul.Popup)` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/impl/XulElement.html#setTooltip\(org.zkoss.zul.Popup\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/impl/XulElement.html#setTooltip(org.zkoss.zul.Popup)))). For example,

```
<zk>
  <image src="/img/earth.png" tooltip="msg"/>

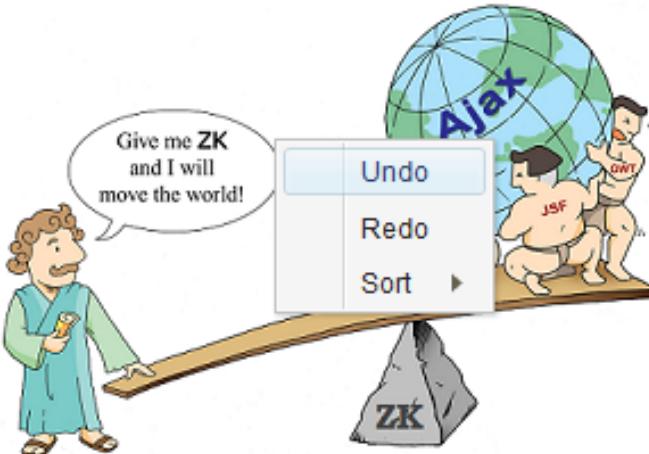
  <menupopup id="msg">
    <menuitem label="Undo"/>
    <menuitem label="Redo"/>
    <menu label="Sort">
      <menupopup>
        <menuitem label="Sort by Name" autocheck="true"/>
        <menuitem label="Sort by Date" autocheck="true"/>
    </menupopup>
  </menupopup>
```

```

</menupopup>
</menu>
</menupopup>
</zk>

```

Then, when the user moves the mouse pointer over the image for a while, the menupopup will show up as shown below.



The time to wait before showing up the tooltip can be configured. Please refer to ZK Configuration Reference for more information.

Context Menus

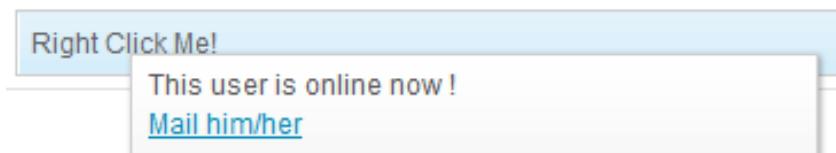
Providing a customized context menu is the same, except it uses the `context` property instead. For example,

```

<zk>
    <listbox>
        <listitem label="Right Click Me!" context="status"/>
    </listbox>

    <popup id="status" width="300px">
        <vlayout>
            This user is online now !
            <a label="Mail him/her"/>
        </vlayout>
    </popup>
</zk>

```



As shown above, you could use Popup (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Popup.html#>) so the context menu is not limited to a menupopup.

Here is another example: context menus versus right clicks.

```
<zk>
    <menupopup id="editPopup">
        <menuitem label="Undo"/>
        <menuitem label="Redo"/>

        <menu label="Sort">
            <menupopup>
                <menuitem label="Sort by Name" autocheck="true"/>
                <menuitem label="Sort by Date" autocheck="true"/>
            </menupopup>
        </menu>
    </menupopup>

    <label value="Right Click Me!" context="editPopup"/>
    <separator bar="true"/>
    <label value="Right Click Me!" onRightClick="alert(self.value)"/>
</zk>
```



Notice that the `menupopup` is not visible until a user right-clicks on a component that is associated with its' ID.

Tip: If you want to disable browser's default context menu, you can set the `context` attribute of a component to a non-existent ID.

The `popup` component is a generic popup and you are able to place any kind of component inside of `popup`. For example,

```
<zk>
    <label value="Right Click Me!" context="any"/>
</zk>

<zk>
    <label value="Right Click Me!" context="any"/>

    <popup id="any" width="300px">
        <vbox>
            It can be anything.
            <toolbarbutton label="ZK" href="http://www.zkoss.org/" />
        </vbox>
    </popup>
</zk>
```

Popups

Providing a customized popup is the same, except it uses the `popup` property instead. For example,

```
<zk>
    <label value="Click Me!" popup="status"/>

    <popup id="status" width="300px">
        <vlayout>
            This user is online now !
            <a label="Mail him/her"/>
        </vlayout>
    </popup>
</zk>
```

Position of Popup

The context-menu/tooltip/popups can be shown by a position from Popup or the location of x and y, you can specify position with the following format:

```
<zk>
    <label value="Click Me!" popup="status, at_pointer"/>
    <label value="Click Me!" popup="status, after_pointer"/>
    <label value="Click Me!" popup="status, position=before_start"/><!-- And other 20 positions -->
    <label value="Click Me!" popup="status, x=30, y=30"/><!-- Fixed positions -->

    <popup id="status" width="300px">
        <vlayout>
            This user is online now !
            <a label="Mail him/her"/>
        </vlayout>
    </popup>
</zk>
```

Check Popup Component for detailed description.

The position is now more customizable and can be done by specifying a formula that executes on client side, for example,

```
<zk>
    <!-- bottom right of mouse pointer -->
    <label value="Click Me!" popup="status, x=(zk.currentPointer[0] + 30), y=(zk.currentPointer[1] + 50)"/>

    <popup id="status" width="300px">
        <vlayout>
            This user is online now !
            <a label="Mail him/her"/>
        </vlayout>
    </popup>
</zk>
```

With `x=(zk.currentPointer[0] + 30), y=(zk.currentPointer[1] + 50)` arguments, the popup position will be on the bottom right of the current mouse pointer.

Note: The parentheses '()' are required to dynamically calculate position on the client side.

Toggle Popup

The context-menu/popup supports toggle type in ZK 7.0.0. Check Popup Component for detailed description.

Override the Reference Component

The reference component of the position can be overridden in ZK 9.5.0. For example, you want the popup to stick to the Textbox, not the triggering button.

```
<zk>
  <inputgroup>
    <textbox id="addr"/>
    <button label="more" popup="pp, ref=addr, position=after_start"/>
  </inputgroup>

  <!-- Your popup -->
  <popup id="pp">
    show something
  </popup>
</zk>
```

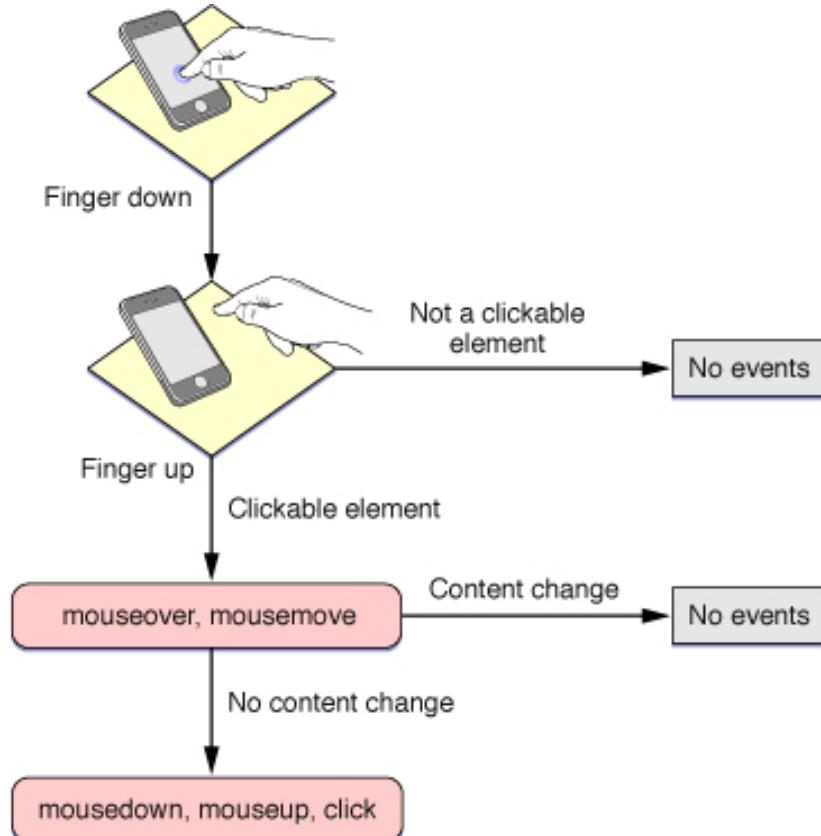
Limitation of iOS

Toolips

Since there is no mouse move event in iOS, the tooltip won't be shown.

Context Menu

ZK Client Engine will simulate the context menu, if the user touches the DOM element associated with the contextmenu property and holds for a while.



Popup

Like onClick, ZK Client Engine simulates the click, if the user touches the DOM element associated with the popup property.

For more information, please refer to Safari Developer Library (http://developer.apple.com/library/safari/documentation/AppleApplications/Reference/SafariWebContent/HandlingEvents/HandlingEvents.html#/apple_ref/doc/uid/TP40006511-SW1).

Version History

Version	Date	Content
5.0.7	May 2011	Context Menus supported with iOS
9.5.0	September 2020	ZK-4551 (https://tracker.zkoss.org/browse/ZK-4551): Support ref override for position of Popup/Tooltip/Context

Keystroke Handling

Keystroke handling is generic. Any component inherited from XulElement^[1] can handle the key event in the same way.

ENTER and ESC

To handle ENTER key pressing, you can listen to the event:

- **onOK** (notice O and K are both in upper case).

To handle ESC key pressing, you can listen to the event:

- **onCancel**

For example:

```
<grid id="form" apply="org.zkoss.reference.developer.uipattern.KeystrokeComposer">
    <rows>
        <row>Username:
            <textbox id="username"/>
        </row>
        <row>Password:
            <textbox id="password" type="password"/>
        </row>
        <row>
            <button label="Login" forward="form.onOK"/>
            <button label="Reset" forward="form.onCancel"/>
        </row>
    </rows>
</grid>
```

Then, you could implement a composer as follows.

```
package org.zkoss.reference.developer.uipattern;

import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.select.SelectorComposer;
import org.zkoss.zk.ui.select.annotation.*;
import org.zkoss.zul.Textbox;

public class KeystrokeComposer extends SelectorComposer<Component> {

    @Wire
```

```

private Textbox username;
@Wire
private Textbox password;

@Listen("onOK = #form")
public void onOK() {
    //handle login
    System.out.println("ok");
}

@Listen("onCancel = #form")
public void onCancel() {
    username.setValue("");
    password.setValue("");
}
}

```

Notice that the `onOK` and `onCancel` events are sent to the nearest ancestor of the component that has the focus. In other words, if you press ENTER in a textbox, then ZK will look up the textbox, its parent, its parent's parent and so on to see if any of them has been registered as a listener for `onOK`. If found, the event is sent to it. If not found, nothing will happen.

Also notice that, if a button gains the focus, ENTER will be intercepted by the browser and interpreted as pressed. For example, if you move the focus to the Reset button and press ENTER, you will receive `onCancel` rather than `onOK` (since `onClick` will be fired and it is converted to `onCancel` because of the `forward` attribute specified).

Control Keys

To handle the control keys, you have to specify the keystrokes you want to handle with `XulElement.setCtrlKeys(java.lang.String)`^[1]. Then, if any child component gains the focus and the user presses a keystroke that matches the combination, the `onCtrlKey` will be sent to the component with an instance of `KeyEvent`^[2].

Like ENTER and ESC, you could specify the listener and the `ctrlKeys` property in one of the ancestors. ZK will search the component having the focus, its parent, its parent's parent and so on to find if any of them specifies the `ctrlKeys` property that matches the keystroke.

For example,

```

<vbox ctrlKeys="@c^a#f10^#f3" onCtrlKey="doSomething(event.getKeyCode())">
    <textbox/>
    <datebox/>
</vbox>

```

As shown, you could use `KeyEvent.getKeyCode()`^[3] to know which key was pressed.

Allowed Control Keys

Key	Syntax	Description
Control	<code>^[?]</code>	<code>[?]</code> can be a~z, 0~9, #[?] , e.g. <code>^k</code> represents Ctrl+k
Alt	<code>@[?]</code>	<code>[?]</code> can be a~z, 0~9, #[?] , e.g. <code>@k</code> represents Alt+k
Shift	<code>\$[?]</code>	<code>[?]</code> can be #[?] . Note: \$a ~ \$z are not supported. e.g. <code>\$#down</code> represents Shift+↓
Mac command(⌘)	<code>%[?]</code>	<code>[?]</code> can be a~z, 0~9, #[?] , e.g. <code>%k</code> represents command+k
Navigation key	<code>#[?]</code>	the supported value of <code>[?]</code> are listed below:
Home	<code>#home</code>	
End	<code>#end</code>	
Insert	<code>#ins</code>	
Delete	<code>#del</code>	
←	<code>#left</code>	
→	<code>#right</code>	
↑	<code>#up</code>	
↓	<code>#down</code>	
PgUp	<code>#pgup</code>	
PgDn	<code>#pgdn</code>	
Backspace	<code>#bak</code>	
function key (F1, F2,... F12)	<code>#f1, #f2, ... #f12</code>	
Tab	<code>#tab</code>	
Space	<code>#space</code>	

Document-level Keystrokes

If you set the library property `org.zkoss.zk.ui.invokeFirstRootForAfterKeyDown.enabled` ^[4] to `true` and there is no widget gaining a focus when an end-user presses a keystroke, ZK will fire a key event to **the first root component that has an onCtrlKey listener**. For example, when visiting the following page, the `div` component will receive the `onOK` event.

```
<div onOK="doSomething(event)" ctrlKeys="^K" onCtrlKey="doSomething(event)">
press enter key or ctrl+k.

<zscript><![CDATA[
public void doSomething(KeyEvent e) {
    Clients.showNotification(e.getKeyCode() + " ");
}
]]></zscript>
</div>
```

In other words, `doSomething()` will be called if a user presses ENTER, even though no widget ever gains the focus.

Nested Components

Keystrokes are propagated up from the widget gaining the focus to the first ancestor widget that handles the keystroke. For example,

```
<div onOK="doFirst()">
  <textbox id="t1"/>
  <div onOK="doSecond()">
    <textbox id="t2"/>
  </div>
</div>
```

Then, `doSecond()` is called if `t2` is the current focus, and `doFirst()` is called if `t1` has the focus.

Key handling and onChange event

When an `onChange` listener alone is registered on a component, `onChange` will be triggered by blur events exclusively.

However, some key events will cause a check for change value and will fire a change event if necessary.

These key events are: `onOK`, `onCancel`, and `onCtrlkeys`. If a listener for any of these events is registered and triggered, an `onChange` event calculation will be triggered, and an `onChange` event will be fired if the value of the control has changed.

Version History

Version	Date	Content
5.0.6	January 2011	Document-level keystroke handling was introduced.
9.5.1	November 2020	Add Tab key support
10.0.0	December 2023	Add Space key support

References

- [1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/impl/XulElement.html#setCtrlKeys\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/impl/XulElement.html#setCtrlKeys(java.lang.String))
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/KeyEvent.html#>
- [3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/KeyEvent.html#getKeyCode\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/KeyEvent.html#getKeyCode())
- [4] https://www.zkoss.org/wiki/ZK_Configuration_Reference/zk.xml/The_Library.Properties/org.zkoss.zk.ui.invokeFirstRootForAfterKeyDown.enabled

Drag and Drop

ZK allows a user to drag particular components around the user interface. For example, dragging an image representing a file onto a tree representing a directory, or dragging a listitem representing a product onto a listbox representing a shopping cart.

A component is droppable, if a user could drop a draggable component to it.

Notice that ZK does not assume any behavior about what should take place after dropping. It is up to application developers to implement the `onDrop` event listener.

If an application doesn't do anything, the dragged component is simply moved back to where it originated from.

The draggable and droppable Properties

With ZK, you could make a component draggable by assigning any value, other than "false", to the draggable property by the use of `HtmlBasedComponent.setDraggable(java.lang.String)` ^[1]. To disable it, assign it with "false".

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

Similarly, you could make a component droppable by assigning "true" to the droppable property by the use of `HtmlBasedComponent.setDroppable(java.lang.String)` ^[2].

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

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

Since the draggable and droppable properties are implemented in `HtmlBasedComponent` ^[3], almost all the components can become draggable or droppable.

The onDrop Event

Once a user has dragged a component and dropped it to another component, the component that the user dropped the component to will be notified by the `onDrop` event. The `onDrop` event is an instance of `DropEvent` ^[4]. By calling `DropEvent.getDragged()` ^[5], you could retrieve what has been dragged (and dropped).

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

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

```
Unique Visitors of ZK:  
<listbox id="src" multiple="true" width="300px">  
    <listhead>  
        <listheader label="Country/Area"/>  
        <listheader align="right" label="Visits"/>  
        <listheader align="right" label="%"/>  
    </listhead>  
    <listitem draggable="true" droppable="true" onDrop="move(event.dragged)">  
        <listcell label="United States"/>  
        <listcell label="5,093"/>  
        <listcell label="19.39%"/>  
    </listitem>  
    <listitem draggable="true" droppable="true" onDrop="move(event.dragged)">
```

```

<listcell label="China"/>
<listcell label="4,274"/>
<listcell label="16.27%"/>
</listitem>
<listitem draggable="true" droppable="true" onDrop="move (event.dragged)">
    <listcell label="France"/>
    <listcell label="1,892"/>
    <listcell label="7.20%"/>
</listitem>
<listitem draggable="true" droppable="true" onDrop="move (event.dragged)">
    <listcell label="Germany"/>
    <listcell label="1,846"/>
    <listcell label="7.03%"/>
</listitem>
<listitem draggable="true" droppable="true" onDrop="move (event.dragged)">
    <listcell label="(other)"/>
    <listcell label="13,162"/>
    <listcell label="50.01%"/>
</listitem>
<listfoot>
    <listfooter label="Total 132"/>
    <listfooter label="26,267"/>
    <listfooter label="100.00%"/>
</listfoot>
</listbox>
<zscript>
    void move (Component dragged) {
        self.parent.insertBefore(dragged, self);
    }
</zscript>

```

Dragging with Multiple Selections

When a user drag-and-drops a list item or a tree item, the selection status of these items won't be changed. Usually only the dragged item is moved, but you can handle all the selected items at once by looking up the set of all selected items as depicted below.

```

public void onDrop(DropEvent evt) {
    Set selected =
    ((Listitem)evt.getDragged()).getListbox().getSelectedItems();
    //then, you can handle the whole set at once
}

```

Notice that the dragged item may not be selected. Thus, you may prefer to change the selection to the dragged item for this case, as shown below.

```

Listitem li = (Listitem)evt.getDragged();
if (li.isSelected()) {
    Set selected =

```

```
((Listitem)evt.getDragged()).getListbox().getSelectedItems();  
    //then, you can handle the whole set at once  
} else {  
    li.setSelected(true);  
    //handle li only  
}
```

Multiple Types of Draggable Components

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

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

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

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

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

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

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

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

Drag-and-Drop Effect Customization

The effects of drag-and-drop can be customized. It requires some client-side programming. Please refer to ZK Client-side Reference/Customization/Drag-and-Drop Effects for more information.

References

- [1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlBasedComponent.html#setDraggable\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlBasedComponent.html#setDraggable(java.lang.String))
- [2] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlBasedComponent.html#setDroppable\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlBasedComponent.html#setDroppable(java.lang.String))
- [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlBasedComponent.html#>
- [4] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/DropEvent.html#>
- [5] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/DropEvent.html#getDragged\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/DropEvent.html#getDragged())

Page Initialization

Sometimes it is helpful to run some code before ZK Loader instantiates any component. For example, check if the user has the authority to access, initialize some data, or prepare some variables for EL expressions.

This can be done easily by implementing Initiator^[1]/InitiatorExt^[2], and then specifying it with the init directive.

```
<?init class="com.foo.MyInitial"?>
```

Initiator and EL

To prepare a variable for EL expression in an initiator, you could store the variable in the page's attributes.

Notice that the provision of variables for EL expression is generally better to be done with VariableResolver^[2] (and then specified it with the variable-resolver directive).

For example, suppose we have a class, CustomerManager, that can be used to load all customers, then we could prepare a variable to store all customers as follows.

```
public class AllCustomerFinder implements Initiator, InitiatorExt {  
  
    @Override  
    public void doInit(Page page, Map args) throws Exception {  
        String name = (String)args.get("name");  
        page.setAttribute(name != null ? name: "customers",  
CustomerManager.findAll());  
    }  
    ...  
}
```

Then, we could use the initiator in a ZUML document as follows.

```
<?init class="my.AllCustomerFinder" name="customers"?>  
  
<listbox id="personList" width="800px" rows="5">  
    <listhead>  
        <listheader label="Name"/>  
        <listheader label="Surname"/>  
        <listheader label="Due Amount"/>  
    </listhead>  
    <listitem value="${each.id}" forEach="${pageScope.customers}">  
        <listcell label="${each.name}"/>  
        <listcell label="${each.surname}"/>  
        <listcell label="${each.due}"/>  
    </listitem>  
</listbox>
```

System-level Initiator

If you have an initiator that shall be invoked for each page, you could register a system-level initiator, rather than specifying it on every page.

It could be done by specifying the initiator you implemented in `WEB-INF/zk.xml` as follows. For more information, please refer to ZK Configuration Reference.

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

Once specified, an instance of the given class will be instantiated for each page (Page^[8]), and then its method will be called as if they are specified in the page (the init directive).

Exception Handling

The initiator can be used to handle the exception when ZK Loader renders a page by implementing `InitiatorExt.doCatch(java.lang.Throwable)`^[3]

Notice that it does not cover the exception thrown in an event listener, which could be handled by the use of `ExecutionCleanup`^[4].

```
import org.zkoss.zk.ui.Page;
import org.zkoss.zk.ui.util.Initiator;
import org.zkoss.zk.ui.util.InitiatorExt;

public class ErrorHandler implements Initiator, InitiatorExt {
    public void doInit(Page page, Map args) throws Exception {
    }
    public void doAfterCompose(Page page) throws Exception { //nothing
to do
    }
    public boolean doCatch(Throwable ex) throws Exception {
        //handle exception here
        return shallIgnore(ex); //return true if the exception is safe
to ignore
    }
    public void doFinally() throws Exception {
        //the finally cleanup
    }
}
```

Version History

Version	Date	Content
5.0.7	April 2011	The system-level initiator was introduced.

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Initiator.html#>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/InitiatorExt.html#>
- [3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/InitiatorExt.html#doCatch\(java.lang.Throwable\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/InitiatorExt.html#doCatch(java.lang.Throwable))
- [4] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ExecutionCleanup.html#>

Forward and Redirect

A Web application jumping from one URL to another is usually caused by the user's click on a hyperlink, such as clicking on a button, toolbarbutton, menuitem and a that is associated with the `href` attribute.

```
<button label="Next" href="next.zul"/>
```

It is done at the client without Java code, so it is efficient. However, you could control it on the server (in Java) too, such that you could redirect it based on some information that is available only at the server.

Redirect to Another URL

Redirecting to another URL is straightforward: pass the URL to `Executions.sendRedirect(java.lang.String)`^[1]. A typical use case is to redirect after authenticating the user's login.

```
if (someCondition()) {
    Executions.sendRedirect("/ready.zul");
```

You could also ask the browser to open another browser window from a given URL by the use of `java.lang.String` `Execution.sendRedirect(java.lang.String, java.lang.String)`^[2].

Redirect When Loading

`Executions.sendRedirect(java.lang.String)`^[1] is designed to be used when serving an AU request (aka., Ajax). If you want to redirect to another page when loading a ZUML document, it is more efficient to call `HttpServletResponse.sendRedirect`^{[3][4]}, such that the browser will handle the redirect for you without running any JavaScript code.

For example,

```
if (!isLoggedIn()) {
    Execution exec = Executions.getCurrent();
    HttpServletResponse response =
        (HttpServletResponse)exec.getNativeResponse();
    response.sendRedirect(response.encodeRedirectURL("/login"));
    //assume there is /login
    exec.setVoided(true); //no need to create UI since redirect will
    take place
}
```

Notice that we invoke `Execution.setVoided(boolean)`^[5] to *void* an execution, such that ZK Loader will abort the evaluation of a ZUML document (if you prefer not to generate any UI when redirecting).

Also notice that the casting to `javax.servlet.http.HttpServletResponse` in the above example does *not* work in a portlet, since the native response is an instance of `javax.portlet.RenderResponse`.

To check whether to redirect can be packed as Initiator^[1], see below for an example:

```
public class AuthenticateInit extends
org.zkoss.zk.ui.util.GenericInitiator {
    public void doInit(Page page, Map args) throws Exception {
        if (!isLoggedIn()) {
            Execution exec = Executions.getCurrent();
            HttpServletResponse response =
(HttpServletResponse)exec.getNativeResponse();

            response.sendRedirect(response.encodeRedirectURL("/login")); //assume
there is /login
            exec.setVoided(true); //no need to create UI since redirect
will take place
        }
    }
}
```

Then, you could specify it in your ZUML document:

```
<?init class="foo.AuthenticateInit"?>
```

-
- [1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#sendRedirect\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#sendRedirect(java.lang.String))
 - [2] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#sendRedirect\(java.lang.String,%20java.lang.String,%20java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#sendRedirect(java.lang.String,%20java.lang.String,%20java.lang.String))
 - [3] <http://download.oracle.com/javaee/1.4/api/javax/servlet/http/HttpServletResponse.html#sendRedirect%28java.lang.String%29>
 - [4] It actually sets the refresh header (http://www.metatags.org/meta_http_equiv_refresh).
 - [5] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#setVoided\(boolean\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#setVoided(boolean))

Forward to Another Page

Sometimes we have to forward to another page. For example, when a user visits a page that requires authorization, we could forward it to a login page^[1].

The simplest way is to use the forward directive:

```
<?forward uri="/login.zul" if="${!foo:isLoggedIn()}"?>
```

where we assume `isLoggedIn` is an EL function that returns whether the user has logged in. For more information, please refer to the Conditional Evaluation section.

You could forward to another page by the use of `Executions.forward(java.lang.String)` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#forward\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#forward(java.lang.String))) too.

Notice that forwarding can be called only when loading a page. You cannot call it when handling an Ajax request (such as when a user clicks a button). For handling an Ajax request, you have to use *redirect* as described in the previous section.

Unlike *redirect*, *forward* does *not* change the URL that the browser knows. Rather, it is purely server-side activity: using another page's content instead of the original one to render the output of the given (and the same) URL.

Navigation and desktop cleanup

The client-side ZK Engine monitors outgoing navigation using the javascript `onBeforeUnload` event. This event is fired when the browser's window is about to unload. From the client engine point of view, unload indicates that the current page will no longer be used. Therefore, a `rmDesktop` command holding the relevant desktop ID is fired to let the server know that server-side objects used in the current page can be discarded.

A screenshot of a browser developer tools Network tab. The URL in the address bar is `zkau?dtid=z_o3w&cmd_0=rmDesktop&opt_0=i`. The 'General' section shows the Request URL as `https://www.zkoss.org/zkdemo/zkau?dtid=z_o3w&cmd_0=rmDesktop&opt_0=i`, Request Method as GET, Status Code as 200 OK, and Referrer Policy as no-referrer-when-downgrade. The 'Query String Parameters' section shows `dtid: z_o3w` and `cmd_0: rmDesktop`, both highlighted with blue boxes.

Usually, `onBeforeUnload` will be triggered by outgoing navigation or by closing a browser tab. However, file download may also cause a browser's window to unload if they are performed in the page's main context. In this case, the initial page still exists after navigation as most browsers will handle file download in a separate download manager without closing the page.

For example, the following `zul` code will trigger a file download by causing navigation to the targeted file. Since the `href` attribute is a valid URL, most browsers will start a navigation workflow (which includes triggering `unload`). Once the target replies with a non-document content (ie a file to download), the browser will interrupt navigation and handle the file while remaining on the previous page.

```
<zkl>
    <a href="../myFile.zip">my file</a>
</zkl>
```

At this point however, the client engine has already triggered `rmDesktop` and the current desktop is no longer available.

A screenshot of a browser developer tools Network tab. The URL in the address bar is `zkau?dtid=z_2rl&cmd_0=rmDesktop&opt_0=i`. A download progress bar for `myFile.zip` is shown.

To avoid this issue, download in ZK should receive a target.

New tab

Using `taget=_blank` will open a new blank browser tab and use it as the target for the URL. This will prevent the ZK page from unloading since no navigation is performed in its context. Since the result of navigating to this url is not a document, the new blank tab will automatically be closed as soon as the download starts.

```
<zkl>
    <a href="../myFile.zip" target="_blank">my file</a>
</zkl>
```

Hidden iframe

To avoid the new tab flickering in the client browser, it is also possible to target a different context in the same page. To do so, the page should contain a hidden iframe, and perform the download through it.

```
<zk>
    <a href="./myFile.zip" target="myHiddenIframe">my file</a>
    <iframe name="myHiddenIframe" visible="false"/>
</zk>
```

Note on outgoing navigation

Outgoing navigation to a different document is usually not an issue since the browser leaves the current page. If the user navigates back to the ZK page, a new desktop will be instantiated and will perform using the normal workflow.

This is only a problem when a navigation action doesn't result in the browser leaving the current ZK page, as the page is unloaded but still exists in the client browser.

[1] In addition to forwarding, we could popup a window to ask them to login, i.e., without leaving the current desktop

File Upload and Download

File Upload

Button^[2], Toolbarbutton^[1] and MenuItem^[2] support file upload out-of-box. In other words, you just need to enable it with the steps below:

1. Specify `upload` attribute with `true` to enable file upload
2. Assign an `onUpload` event listener to the component^[3]

For example:

```
<zk>
    <zscript>
        void upload(UploadEvent event) {
            org.zkoss.util.media.Media media = event.getMedia();
            if (media instanceof org.zkoss.image.Image) {
                org.zkoss.zul.Image image = new
org.zkoss.zul.Image();
                image.setContent((org.zkoss.image.Image) media);
                image.setParent(pics);
            } else {
                Messagebox.show("Not an image: "+media, "Error",
Messagebox.OK, Messagebox.ERROR);
            }
        }
    </zscript>
    <button label="Upload" upload="true" onUpload="upload(event)" />
    <vlayout id="pics" />
</zk>
```

You can control the maximal allowed number of files, the maximal allowed size, and other options. Please refer to org.zkoss.zul.Button.setUpload() [4].

```
<menupopup>
    <menuitem label="Upload" upload="true,maxsize=-1,native"/>
</menupopup>
```

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Toolbarbutton.html#>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/MenuItem.html#>
- [3] If you enabled the use of event threads, you could use Button (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Button.html#>) and related. For more information, please refer to the Event Thread section.
- [4] <https://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Button.html#setUpload-java.lang.String->

Open File Upload Dialog

If you want the file upload on another component rather than those components above (Button, Toolbarbutton, Menu), you can call FileUpload.get() (<https://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Fileupload.html#get-->) in an event listener.

Here is an example:

```
<a iconClass="z-icon-upload" ...
    apply="org.zkoss.reference.developer.uipattern.UploadComposer"/>
```

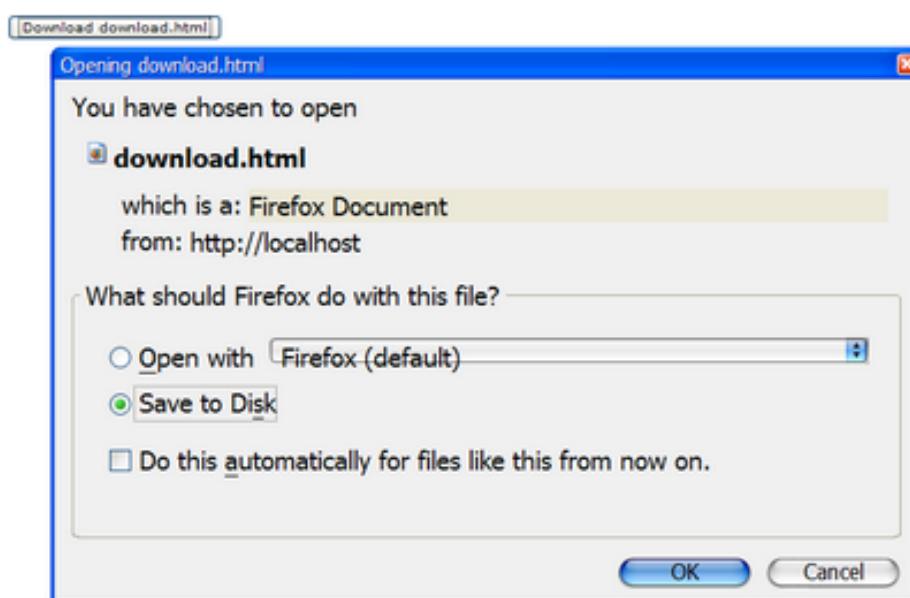
```
public class UploadComposer extends SelectorComposer {

    @Listen(Events.ON_CLICK + "= a")
    public void handleUpload(MouseEvent e) {
        Fileupload.get(1, new EventListener<UploadEvent>() {
            public void onEvent(UploadEvent event) throws Exception {
                Media[] medias = event.getMedias();
                System.out.println("upload " + medias[0].getName());
            }
        });
    }
}
```

File Download

Filedownload provides a set of utilities to prompt a user to download a file from the server to a local folder. For example,

```
<button label="Download a file" onClick='Filedownload.save("~/zklogo.png", null);'>
```



The file could be a static resource, an input stream, a file, a URL and others. Please refer to ZK Component Reference and Filedownload (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Filedownload.html#>) for more information.

File Download Link

Clicking a file download link terminates client engine to work, please refer to [ZK_Component_Reference/Essential_Components/A#File_Download_Link](#) for correct usages.

Browser Information and Control

To retrieve the information about the client, you can register an event listener for the `onClientInfo` event to a root component. To control the behavior of the client, you can use the utilities in the `Clients`^[1] class.

Browser Information

Sometimes an application needs to know the client's information, such as time zone. Then, you can add an event listener for the `onClientInfo` event. Once the event is added, the client will send back an instance of the `ClientInfoEvent`^[2] class, from which you can retrieve the information of the client.

For example,

```
<grid onClientInfo="onClientInfo(event)">
    <rows>
        <row>Time Zone <label id="tm"/></row>
        <row>Screen <label id="scrn"/></row>
    </rows>

    <zscript>
        void onClientInfo(ClientInfoEvent evt) {
            tm.setValue(evt.getTimeZone().toString());
            scrn.setValue(
                evt.getWidth() + "x" + evt.getHeight() + "x" + evt.getColorDepth());
        }
    </zscript>
</grid>
```

Notice that the `onClientInfo` event is meaningful only to the root component (aka., a component without any parent).

The client information is not stored by ZK, so you have to store it manually if necessary. Since a session is associated with the same client, you can store the client info in the session's attribute.

For example, we could use it to control the default time zone (For more information about time zone, please refer to the Time Zone section).

```
session.setAttribute("org.zkoss.web.preferred.timezone",
event.getTimeZone());
```

Notice that the `onClientInfo` event is sent from the client after the UI is rendered at the client. Thus, if some of your component's data depends on the client's info, say, screen resolution, it is better to handle it (say, adjust UI's size) when the `onClientInfo` event is received.

If it is, though rarely, too late (i.e., it has to be done at the beginning), you could ask the client to re-send the request again with `Executions.sendRedirect(java.lang.String)`^[1]. For example,

```
import org.zkoss.util.TimeZones;
...
if (!TimeZones.getCurrent().equals(event.getTimeZone())) {
    session.setAttribute("org.zkoss.web.preferred.timezone",
event.getTimeZone()); //update to the session
```

```
    Executions.sendRedirect(null); //ask to re-send (i.e., redo)
}
```

Browser Control

Clients^[3] has utilities to control the client's visual presentation (more precisely, the browser window), such as printing, submitting, resizing and so on. For example, you can scroll the browser window (aka., the desktop) as follows.

```
Clients.scrollBy(100, 0);
```

Here we describe some special controls that are worth noticing. For complete functionality, please refer to Clients^[3].

Warn Users When Leaving a Page

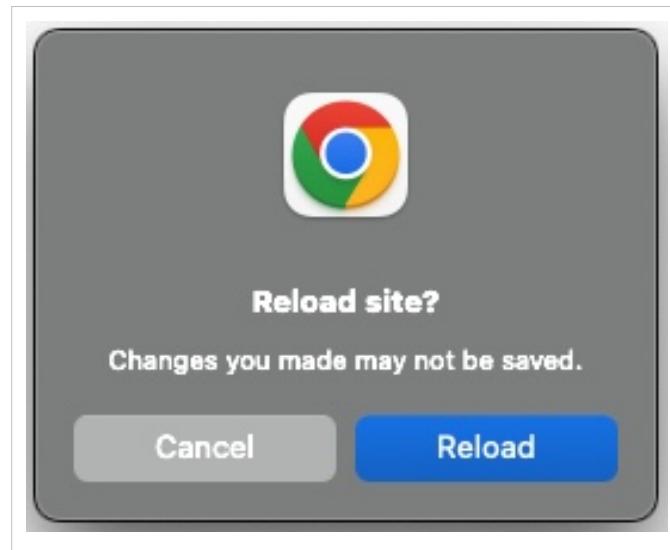
If you want to confirm a user when they try to reload, close, or leave the current page to another URL, you can call Clients.confirmClose(java.lang.String)^[4].

For example, when a user is composing a mail that is not sent or saved yet.

```
if (mail.isDirty()) {
    Clients.confirmClose("any string");
} else {
    Clients.confirmClose(null); //reset. no more confirmation.
}
```

- Line 2: All modern browsers now don't support showing a custom message^[5]. So just pass a non-null string as a parameter to activate the confirmation dialog.

After activating, when the user tries to leave a page, a browser will show a confirmation dialog (each browser has its own default dialog with a default message):



To disable the confirmation dialog, just invoke Clients.confirmClose(java.lang.String)^[4] with null.

ZK cannot determine what's unsaved or dirty, you have to implement it by yourselves. Then call this API in a proper timing.

Notify User Component under Processing

Sometimes a request may take a long time to process, and it is better to notify the user that it is under processing. It can be done by the use of `java.lang.String) Clients.showBusy(org.zkoss.zk.ui.Component, java.lang.String)`^[6] if only a component is not accessible, or `Clients.showBusy(java.lang.String)`^[7] if the whole browser is not accessible. For example,

```
showBusy(grid, "Loading data...");
```

After the loading is completed, you could invoke `Clients.clearBusy(org.zkoss.zk.ui.Component)`^[8] (or `Clients.clearBusy()`^[9]) to clean it up. For more information, please refer to the Use Echo Events section.

Log the Message to Browser

In addition to logging messages to the console, you could log the messages to the browser for debugging by the use of `Clients.log(java.lang.String)`^[10]. For example,

```
//in Java
void doSomething() {
    Clients.log("doSomething called");
}
```

Control in JavaScript

If you are familiar with JavaScript, you could have more control by sending any JavaScript code to the client for evaluation. It can be done by preparing the JavaScript code in `AuInvoke`^[11] or `AuScript`^[12], and then send back to the client by calling `Clients.response(org.zkoss.zk.au.AuResponse)`^[13].

For example, we could use `AuScript`^[12] to inject any code, while `AuInvoke`^[11] is better if you want to invoke a function of a client-side widget.

```
Clients.response(new AuScript(null, "alert('Hello, Client')"));
```

For client-side API, please refer to JavaScript API^[14].

Browser Page Visibility State

In order to develop power and CPU efficient web applications, W3C publishes a specification named `Page Visibility`^[15] in HTML 5 which defines a means for site developers to programmatically determine the current visibility state of the page. In this specification, there are two attributes defined: `hidden` and `visibilityState`, where `hidden` is a boolean value representing whether the current page is visible or not and `visibilityState` represents that the current page has four states: `hidden`, `visible`, `prerender`, and `unloaded`.

To get the two attributes, you can add an event listener for the `onVisibilityChange` event. Once the event is added, the client will send back an instance of the `VisibilityChangeEvent`^[16] class, from which you can retrieve the page visibility state of the current page.

```
<zk>
<label>
Open/Change to another tab in the browser and go back this tab
</label>
<window title="window" border="normal" onVisibilityChange="onVisibleChange(event)">
```

```

<label id="lbl" />
<zscript><! [CDATA[
    import org.zkoss.zk.ui.event.VisibilityChangeEvent;
    void onVisibleChange(VisibilityChangeEvent event) {
        if
("visible".equals(event.getVisibilityState()) || !event.isHidden()) {
            lbl.setValue("Welcome back");
        }
    }
]]></zscript>
</window>
</zk>
```

Notice that the `onVisibilityChange` event is meaningful only to the root component (aka., a component without any parent) and the browsers that support this HTML 5 API.

Version History

Version	Date	Content
5.0.8	June, 2010	Clients.log(java.lang.String) ^[10] was introduced.
6.5.1	November, 2012	VisibilityChangeEvent ^[17] was introduced.

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/ClientInfoEvent.html#>
- [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/ui/util/Clients.html#>
- [4] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#confirmClose\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#confirmClose(java.lang.String))
- [5] https://developer.mozilla.org/en-US/docs/Web/API/Window/beforeunload_event#compatibility_notes
- [6] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#showBusy\(org.zkoss.zk.ui.Component](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#showBusy(org.zkoss.zk.ui.Component),
- [7] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#showBusy\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#showBusy(java.lang.String))
- [8] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#clearBusy\(org.zkoss.zk.ui.Component\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#clearBusy(org.zkoss.zk.ui.Component)
- [9] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#clearBusy\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#clearBusy())
- [10] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#log\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#log(java.lang.String))
- [11] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/au/out/AuInvoke.html#>
- [12] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/au/out/AuScript.html#>
- [13] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#response\(org.zkoss.zk.au.AuResponse\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#response(org.zkoss.zk.au.AuResponse))
- [14] <http://www.zkoss.org/javadoc/latest/jsdoc/>
- [15] <http://www.w3.org/TR/page-visibility/>
- [16] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/VisibilityChangeEvent.html#>
- [17] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/VisibilityChangeEvent.html#>

Browser History Management

History Management with Bookmark

In traditional multi-page Web applications, users usually use the BACK and FORWARD button to surf around multiple pages, and bookmark them for later use. With ZK, you still can use multiple pages to represent different sets of features and information, as you did in traditional Web applications.

However, it is more common for ZK applications to represent a lot of features in one desktop, which usually requires multiple Web pages in a traditional Web application. To make user's surfing easier, ZK supports the browser's history management that enables ZK applications to manage browser's history simply on the server.

The concept is simple. For each state of a desktop, you could add a so-called bookmark^[1] to the browser's history. Then, the user can use the BACK and FORWARD button of the browser to switch around different bookmarks. The change of books will be sent back to the server called `onBookmarkChange`, and your application can switch to the corresponding accordingly.

From application's viewpoint, it takes two steps to manage the browser's history:

1. Add a bookmark to the browser's history for each of the visited states of your desktop.
2. Listen to the `onBookmarkChange` event for bookmark change, and switch the state accordingly.

[1] Each bookmark is an arbitrary string added to the browser's history.

Add Bookmarks to Browser's History

Your application has to decide what the appropriate states are to add to the browser's history. For example, in a multi-step operation, each step is a good candidate of states for adding to a browser's history, such that the user can switch around these steps by pressing the BACK or FORWARD buttons.

Once you decide when to add a state to the browser's history, you can simply invoke `Desktop.setBookmark(java.lang.String)` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Desktop.html#setBookmark\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Desktop.html#setBookmark(java.lang.String))). Notice that it is *not* the bookmarks that users add to the browser (aka., My Favorites in Internet Explorer).

For example, assume you want to bookmark the state when the Next button is clicked, then you do as follows.

```
<button label="Next" onClick='desktop.setBookmark("Step-2")' />
```

If you look carefully at the URL, you will find ZK appends `#Step-2` to the URL.

If you press the BACK button, you will see as follows.



http://localhost/zkdemo/test/bookmark.zul#Step-2

Listen to onBookmarkChange and Change the State Accordingly

After adding a bookmark to the browser's history, users can then surf among these bookmarks such as pressing the BACK button to return to the previous bookmark. When the bookmark is changed, ZK will notify the application by broadcasting the `onBookmarkChange` event (an instance of the `BookmarkEvent` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/BookmarkEvent.html#>) class) to all root components in the desktop.

Unlike traditional multi-page Web applications, you have to change the desktop's state on the server manually, when `onBookmarkChange` is received. ZK does nothing to allow an application to set a bookmark and notify for the bookmark change. It is the application developer's job to manipulate the desktop to reflect the state that a bookmark has represented.

To listen to the `onBookmarkChange` event, you can add an event listener to any pages of the desktop, or to any of its root components.

```
<window onBookmarkChange="goto(event.bookmark)">
    <zscript>
        void goto(String bookmark) {
            if ("Step-2".equals(bookmark)) {
                ...//create components for Step 2
            } else { //empty bookmark
                ...//create components for Step 1
            }
    </zscript>
</window>
```

Like handling any other events, you can manipulate the UI any way you want, when the `onBookmarkChange` event is received. It is totally up to you.

A typical approach is to use one of the `createComponents` methods of the `Executions` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#>) class. In other words, you could represent each state with one ZUML page, and then use `createComponents` to create all components in it when `onBookmarkChange` is received.

```
if ("Step-2".equals(bookmark)) {
    //1. Remove components, if any, representing the previous state
    try {
        self.getFellow("replacable").detach();
    } catch (ComponentNotFoundException ex) {
        //not created yet
    }

    //2. Create components belonging to Step 2
    Executions.createComponents("/bk/step2.zul", self, null);
}
```

Example

In this example, we bookmark each tab selection.

```
<window id="wnd" title="Bookmark Demo" width="400px" border="normal">
<zscript>
    page.addEventListener(Events.ON_BOOKMARK_CHANGE,
        new EventListener() {
            public void onEvent(Event event) throws UiException {
                try {
                    wnd.getFellow(wnd.desktop.bookmark).setSelected(true);
                } catch (ComponentNotFoundException ex) {
                    tab1.setSelected(true);
                }
            }
        });
</zscript>

<tabbox id="tbox" width="100%" onSelect="desktop.bookmark = self.selectedTab.id">
    <tabs>
        <tab id="tab1" label="Tab 1"/>
        <tab id="tab2" label="Tab 2"/>
        <tab id="tab3" label="Tab 3"/>
    </tabs>
    <tabpanels>
        <tabpanel>This is panel 1</tabpanel>
        <tabpanel>This is panel 2</tabpanel>
        <tabpanel>This is panel 3</tabpanel>
    </tabpanels>
</tabbox>
</window>
```

Bookmarking with iframe

If a page contains one or more `iframe` components, it is sometimes better to bookmark the status of the `iframe` components too. For example, when the contained `iframe` is navigated to another URL, you might want to change the bookmark of the page (the container), such that you can restore to the `iframe` to the right content. To do this, you have to listen to the `onURICHange` event as follows.

```
<window onURICHange="desktop.bookmark = storeURI(event.getTarget(), event.getURI())">
    <iframe src="${uri_depends_on_bookmark}" forward="onURICHange"/>
</window>
```

The `onURICHange` event is sent as an instance of `URIEvent` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/URIEvent.html#>).

Notice that the `onURICHange` event is sent only if the `iframe` contains another ZK page. If it contains a non-ZK page, you have to handle it manually. Please refer to ZK Component Reference: `iframe` for more information.

Session Timeout Management

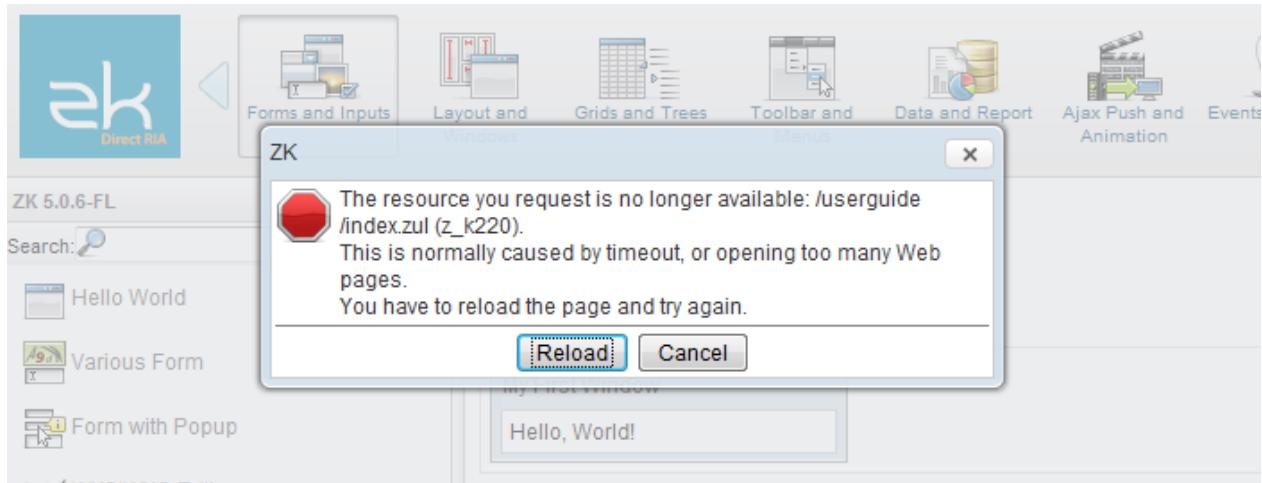
After a session times out, all desktops and UI objects it belongs are removed. If a user keeps accessing the desktop that no longer exists, ZK will prompt the user for the session-timeout situation. ZK supports several ways to prompt the user for session timeout:

- Show a message
- Redirect to another page
- Totally control by running JavaScript code

You could pick one depending on your application requirement. In addition, you could configure your application to enforce the user prompting to take place, without waiting for the user's activity. It is called automatic timeout.

Show a Message

By default, a message is shown to prompt the user and prevent from further accessing as depicted below.



Custom Message

You could show a custom message by specifying `timeout-message` in `WEB-INF/zk.xml`. For example,

```
<session-config>
    <device-type>ajax</device-type>
    <timeout-message>Session timeout. Please reload.</timeout-message>
</session-config>
```

Internationalization

If you want to specify a Locale-dependent message, you could specify the key and prefix it with `label:` as follows.

```
<session-config>
    <device-type>ajax</device-type>
    <timeout-message>label:timeout</timeout-message>
</session-config>
```

Then, you have to prepare the zk-label properties files as described in the Labels section.

```
#zk-label.properties
timeout={
Session timeout.
(multi-line is allowed)
}
```

Redirect to Another Page

Sometimes it is better to redirect to another page that gives users a more complete description and guides them to the other resources, or asks them to login again. You can specify the target URI, that you want to redirect users to when timeout, with the timeout-uri element in WEB-INF/zk.xml. For example, the target URI is /timeout.zul and then you can add the following lines to zk.xml.

```
<session-config>
  <device-type>ajax</device-type>
  <timeout-uri>/timeout.zul</timeout-uri>
</session-config>
```

In addition to WEB-INF/zk.xml, you could change the redirected URI manually as follows.

```
Devices.setTimeoutURI("ajax", "/timeout.zul");
```

About Device: A device represents the client device, such as Ajax browsers and Android devices. Each desktop is associated with one device, and vice versa.

If you prefer to reload the page instead of redirecting to other URI, you can specify an empty URI as follows.

```
<session-config>
  <device-type>ajax</device-type>
  <timeout-uri></timeout-uri>
</session-config>
```

Total Control in JavaScript

If you want more amazing effect, you could provide some JavaScript code and configure ZK to run it if timeout. For example, our demo ^[1] shows up a message on the top of window with some animation, and then automatically reloads if it detects any mouse move (it means the user is back).

For example, if you have a function called `foo.timeout` to handle the timeout effect, then you could configure WEB-INF/zk.xml as follows.

```
<session-config>
  <device-type>ajax</device-type>
  <automatic-timeout>true</automatic-timeout>
  <timeout-message>script:<! [CDATA[foo.timeout('Session Timeout');]]></timeout-message>
</session-config>
```

The code depends on the client. For Ajax devices, it has to be JavaScript.

Automatic Timeout

By default, the session-timeout mechanism is triggered only if the client sends back a request (such as clicking on a button). If you prefer to prompt the user even if it doesn't do anything, you could specify the automatic-timeout element in WEB-INF/zk.xml as follows.

```
<session-config>
    <device-type>ajax</device-type>
    <automatic-timeout/>
</session-config>
```

Then, ZK Client will trigger the session-time mechanism (showing a message, redirecting to another page, or running some JavaScript code).

Page-level Automatic Timeout

If you want to specify whether to automatically timeout for particular pages, you can use the page directive.

Moreover, it is better to turn off the automatic timeout for the timeout page you want to redirect to (if the page is a ZUML page). For example,

```
<!-- my timeout page -->
<?page automaticTimeout="false"?>
...
```

Never Timeout

Though not recommended, you could prevent the session from timeout by making a "keep-alive" timer, such that the desktop keeps alive until the user surfs away.

To do that, you first configure WEB/zk.xml as follows.

```
<session-config>
    <timer-keep-alive>true</timer-keep-alive>
</session-config>
```

and create a timer in your ZUL page:

```
<timer id="timerKeepAliveSession" repeats="true" delay="600000" onTimer="" />
```

This will prevent the session to time out when the ZUL page is opened in the browser. The session still timeouts when the user has navigated the browser away. The delay (600000 is 10 minutes) shall be as long as possible but smaller than your session timeout.

The timer-keep-alive element is used to specify whether the session shall consider timer as a normal request. If it is considered as a normal request, the session timeout mechanism will be restarted when it is received. Otherwise, the timer, by default, won't restart the timeout mechanism.

Version History

Version	Date	Content
5.0.5	October 2010	The support of Custom Message and JavaScript was introduced.

References

[1] <http://www.zkoss.org/zkdemo>

Error Handling

Here we describe how to handle errors. An error is caused by an exception that is not caught by the application. An exception might be thrown in two situations:

1. when loading a ZUML document
2. when serving an AU request (aka, an Ajax request).

To handle them both, you need to configure them in different places.

Error Handling When Loading ZUML Documents

If an uncaught exception is thrown when loading a ZUML document, it is handled directly by the Web server. In other words, the handling is no different from other servlets.

By default, the Web server displays an error page showing the error message and stack trace. For example,

```
HTTP Status 500 -

type Exception report
message
description: The server encountered an internal error () that prevented it from fulfilling this request.
exception
com.potix.zk.ui.UiException: Recursive import: /test/import.zul
    com.potix.zk.ui.metainfo.Parser.parse(Parser.java:200)
    com.potix.zk.ui.metainfo.Parser.parse(Parser.java:90)
    com.potix.zk.ui.metainfo.PageDefinitions$MyLoader.parse(PageDefinitions.java:186)
    com.potix.web.util.resource.ResourceLoader.load(ResourceLoader.java:94)
    com.potix.util.resource.ResourceCache$Info.load(ResourceCache.java:223)
    com.potix.util.resource.ResourceCache$Info.<init>(ResourceCache.java:197)
    com.potix.util.resource.ResourceCache.get(ResourceCache.java:136)
```

You can customize the error handling by specifying the error page in WEB-INF/web.xml as follows^[1].

Note: When exceptions are thrown during the ZK UI Lifecycle they are wrapped into a UiException^[2]. If you want to handle your own exceptions you can implement the Expectable^[3] on your exception type. Exceptions implementing this interface will not be wrapped and can be handled using the <exception-type> element directly.

Note: If the exception you want to handle is a checked exception, it must extend ServletException or IOException. So the Web container can handle them directly in doGet or doPost method.

```
<!-- WEB-INF/web.xml -->
<error-page>
    <exception-type>java.lang.Throwable</exception-type>
    <location>/WEB-INF/sys/error.zul</location>
</error-page>
```

Then, when an error occurs on loading a page, the Web server forwards the error page you specified, /error/error.zul. Upon forwarding, the Web server passes a set of request attributes to the error page to describe what happens. These attributes are as follows.

Request Attribute	Type
javax.servlet.error.status_code	java.lang.Integer
javax.servlet.error.exception_type	java.lang.Class
javax.servlet.error.message	java.lang.String
javax.servlet.error.exception	java.lang.Throwable
javax.servlet.error.request_uri	java.lang.String
javax.servlet.error.servlet_name	java.lang.String

Then, in the error page, you can display your custom information by the use of these attributes. For example,

```
<window title="Error ${requestScope['javax.servlet.error.status_code']} ">
    Cause: ${requestScope['javax.servlet.error.message']}
</window>
```

Tips:

- The error page can be any kind of servlets. In addition to ZUML, you can use JSP or whatever servlet you prefer.
- From java code the request attributes are accessible via Execution.getAttribute(java.lang.String)^[4] or from the requestScope (implicit object)^[5].

```
public class ErrorHandlingComposer extends SelectorComposer<Component> {

    @WireVariable
    private Map<String, Object> requestScope;

    @Override
    public void doAfterCompose(Component comp) throws Exception {

        //via execution.getAttribute()
        Execution execution = Executions.getCurrent();
        Exception ex1 = (Exception)
execution.getAttribute("javax.servlet.error.exception");

        //via requestScope map
        Exception ex2 = (Exception)
requestScope.get("javax.servlet.error.exception");
    }
}
```

-
- [1] Please refer to Chapter 10.9 of Java Servlet Specification (http://download.oracle.com/otn-pub/jcp/servlet-3.0-fr-eval-oth-JSpec/servlet-3_0-final-spec.pdf) for more details.
 - [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/UiException.html#>
 - [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/lang/Expectable.html#>
 - [4] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#getAttribute\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#getAttribute(java.lang.String))
 - [5] [https://www.zkoss.org/wiki/ZUML_Reference/EL_Expressions/Implicit_Objects_\(Predefined_Variables\)/requestScope](https://www.zkoss.org/wiki/ZUML_Reference/EL_Expressions/Implicit_Objects_(Predefined_Variables)/requestScope)

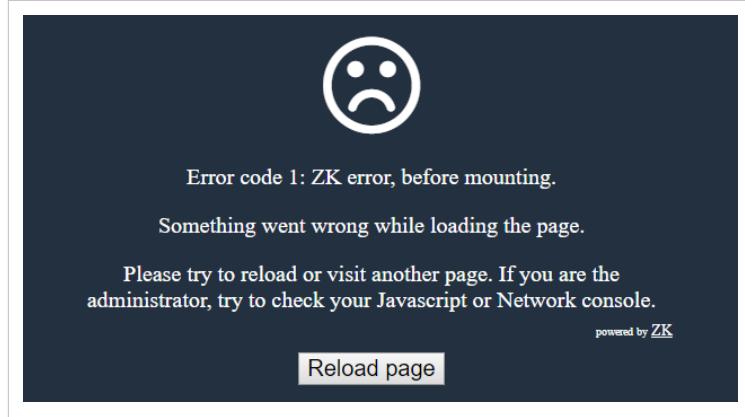
Error Handling when the Client Engine crashes

In rare cases, the client engine stops working before even the error handling is initialized (e.g. when ZK's core scripts, e.g. zk.wpd, fail to download). In those cases, the configured error handler can't be called, and ZK falls back to a very basic error handling.

If the client engine didn't initialize within a configurable timeout, it will display a generic error message like the screenshot to the right. When this occurs, the connection to ZK is usually broken. So you can't report errors to the server via ZK's Ajax engine. As the error details are usually visible in the browser's console, it's useful to instruct users to report the errors manually or automatically extract and send them to an error handling service that is accessible at that time (not part of ZK).

Please check the list of error codes. You can configure both the timeout and the error message presented to users with the elements below:

- <init-crash-script>
- <init-crash-timeout>



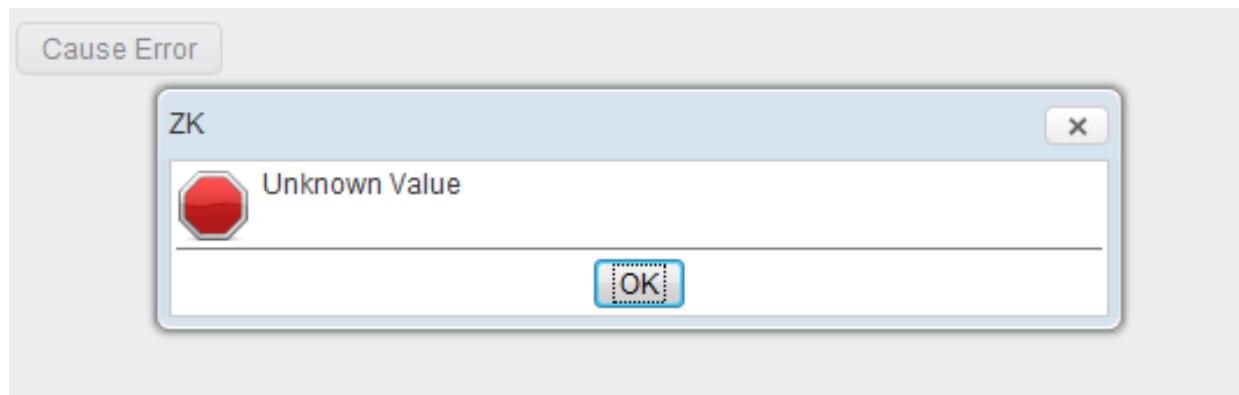
Handling Errors Thrown in Event Listeners

If an uncaught exception is thrown by an event listener or a command method in a ViewModel (aka. when serving an AU request), it is handled by the ZK Update Engine. By default, it simply shows an error message to indicate the error.

For example, suppose we have the following code:

```
<button label="Cause Error" onClick='throw new NullPointerException("Unknown Value")' />
```

Then, if you click the button, the following error message will be shown.



Configure Error Handling Page

You can customize the error handling by specifying the error page in `WEB-INF/zk.xml` as described in ZK Configuration Reference. For example,

```
<!-- zk.xml -->
<error-page>
    <exception-type>java.lang.Throwable</exception-type>
    <location>/WEB-INF/sys/error.zul</location>
</error-page>
```

Then, when an error occurs in an event listener, the ZK Update Engine will show the page you specified.

Like error handling in loading a ZUML page, you can specify multiple `<error-page>`. Each of them is associated with a different exception type, specified in `<exception-type>`. When an error occurs, ZK will search the error pages one-by-one until the exception type matches.

Order Matters

The order to handle a thrown exception according to its type is based on the `<error-page>`'s declaration sequence in `zk.xml`.

Get Error Information

In addition, ZK passes a set of request(Execution) attributes to the error page to describe what happens. These attributes are as follows:

Request Attribute Name	Type	Content
<code>javax.servlet.error.exception_type</code>	<code>java.lang.Class</code>	the thrown error object's class i.e. return <code>Throwable.getClass</code>
<code>javax.servlet.error.message</code>	<code>java.lang.String</code>	the all thrown error messages combined
<code>javax.servlet.error.exception</code>	<code>java.lang.Throwable</code>	the thrown error object
<code>javax.servlet.error.status_code</code>	<code>java.lang.Integer</code>	500
<code>javax.servlet.error.error_page</code>	<code>java.lang.String</code>	the error handling page URL configured in <code>zk.xml</code>

Besides, the error page is created on the same desktop that causes the error, so you can retrieve the relevant information from the desktop e.g. the page URL, `getPage().getRequestPath()`.

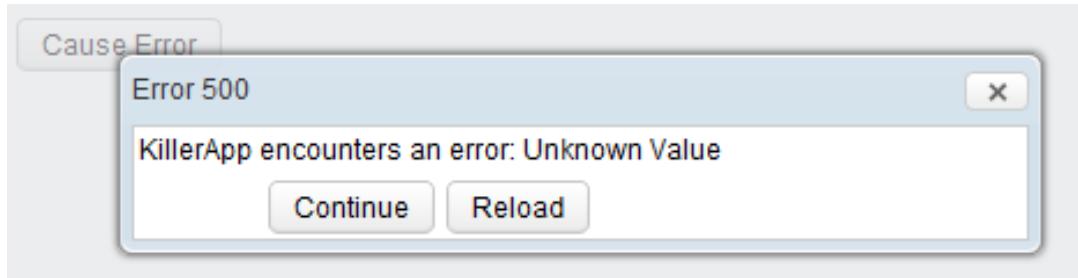
For example, you can specify the following content as the error page.

```
<window title="Error ${requestScope['javax.servlet.error.status_code']} ">
    width="50%" border="normal" mode="modal" closable="true">
        <vlayout>
            KillerApp encounters an error:
            ${requestScope['javax.servlet.error.message']}
            <hlayout style="margin-left:auto; margin-right:auto">
                <button label="Continue" onClick="spaceOwner.detach()"/>
                <button label="Reload" onClick="Executions.sendRedirect(null)"/>
            </hlayout>
        </vlayout>

        <!-- optional: record the error for improving the app -->
        <zscript>
```

```
org.zkoss.util.logging.Log.lookup("Fatal").error(  
    requestScope.get("javax.servlet.error.exception"));  
</zscript>  
</window>
```

Then, when the button is clicked, the following will be shown.



Handling a Custom Exception

By default, ZK will wrap your custom exception with `UiException` or `OperationException`. If you want to handle your custom exception, `YourException`, specifically on a specific error page upon its type. Your custom exception class needs to extend specific classes to avoid being wrapped:

For unchecked:

```
public class YourException extends java.lang.RuntimeException{...}
```

Ref: <http://tracker.zkoss.org/browse/ZK-2638>

For checked:

```
public class YourException extends javax.servlet.ServletException{...}
```

or

```
public class YourException extends java.io.IOException{...}
```

Ref: <http://tracker.zkoss.org/browse/ZK-3679>

Because `HttpServlet` only throws these 2 checked exceptions (`ServletException`, `IOException`) above (<https://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServlet.html#doGet-javax.servlet.http.HttpServletRequest->).

Actions and Effects

The client-side action (CSA) is used to control how to perform an action at the client. Typical use is to control the effect of showing or hiding a widget. For example, with CSA, you could use the so-called *slide-down* effect to display a widget.

It is a generic feature available to `HtmlBasedComponent`^[3], so you could apply it to almost all widgets.

CSA allows the developer to control some actions without JavaScript. If you want to have the full control (and are OK to write some JavaScript code), please refer to ZK Client-side Reference for the complete control of the client-side behavior.

How to Apply Client-side Actions

To apply the client-side action to a widget, you have to assign a value to the `action` property (`HtmlBasedComponent.setAction(java.lang.String)`^[1]). The syntax is as follows.

```
action="action-name1: effect1; action-name2: effect2"
```

The action name (e.g., `action1`) has to be one of the predefined names, such as `show` and `hide`. The action effect (e.g., `effect1`) has to be one of the predefined effects, such as `slideDown` and `slideUp`.

For example, we could use the *slide-down* effect to display a window as follows^[2].

```
<zk>
    <button label="Show a modal window" onClick="wnd.doModal()" />
    <window id="wnd" title="Modal" border="normal" width="300px"
        action="show: slideDown" visible="false">
        This is a modal window.
    </window>
</zk>
```

In addition, you could specify additional options by enclosing them with the parentheses as follows.

```
<div action="show: slideDown({duration: 1000}); hide: slideUp({duration: 300})">
    ...
</div>
```

which specifies the duration of sliding down is 100 milliseconds, and the duration of sliding up is 300 milliseconds.

Security Note: the options is actually a JavaScript object (i.e., a map, `Map`^[3]), and ZK passes whatever is being specified to the client for evaluation. Thus, if you allow the user to specify the effect, you shall encode it first to avoid cross-site scripting.

-
- [1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlBasedComponent.html#setAction\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlBasedComponent.html#setAction(java.lang.String))
 [2] If you are using the effects with a modal window, it is important to specify the width. Otherwise, the calculation of the position might be wrong.
 [3] http://www.zkoss.org/javadoc/latest/jsdoc/_global_/Map.html#

Predefined Actions

Here is a list of predefined actions.

Name	Description
show	The show action is used to display a widget (making a widget visible). When a visible widget is attached to a page, the <code>show</code> action will take place too.
hide	The hide action is used to hide a widget (making a widget invisible). When a visible widget is detached from a page, the <code>hide</code> action will take place too.
invalidate	The invalidate action is invoked when a visible widget is invalidated, i.e., when <code>Component.invalidate()</code> (http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#invalidate()) is called. Example, <code>action="invalidate: slideDown"</code> .

Predefined Effects

Here is a list of predefined effects.

Name	Description
slideDown	Slides down to display this widget (making a widget visible).
slideUp	Slides up to hide this widget (making a widget invisible).
slideIn	Slides in to display this widget (making a widget visible).
slideOut	Slides out to hide this widget (making a widget invisible).

Predefined Options for Effects

Option Name	Acceptable Values	Description
anchor	t, b, l, r	The 4 values represent the direction to slide: top to bottom , bottom to top , left to right , right to left respectively.
duration	a number of milliseconds	The time to slide up/down/in/out a widget. The larger the time, the slower the widget slides.

Custom Actions

If you want to take some actions other than the predefined actions listed above, you have to override the corresponding method at the client. For example, suppose you'd like to change the color when a label's value (`Label.setValue(java.lang.String)` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Label.html#setValue\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Label.html#setValue(java.lang.String))))) is changed. Then, you could do as follows:

```
<label id="inf2">
  <attribute w:name="setValue">
    function (value, fromServer) {
      this.$setValue(value, fromServer);
      if (this.desktop) {
```

```
        this._red = !this._red;
        this.setStyle('background:'+(this._red ?
'red':'green')) ;
    }
}
</attribute>
</label>
```

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

Custom Effects

For adding your custom effects, please refer to ZK Client-side Reference: Customization: Actions and Effects for details.

Notes for Upgrading from ZK 3

They are both called Client-side Actions, but they are different and you have to rewrite them to make them work under ZK 5:

1. The action names were changed and the support is limited to `show` and `hide` (while ZK 3 supports any `onxxxx`).
2. The action operation must be the name of one of the methods defined in Actions (<http://www.zkoss.org/javadoc/latest/jsdoc/zk/eff/Actions.html#>) (while ZK 3 is the JavaScript code).
3. It is part of `HtmlBasedComponent` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlBasedComponent.html#>) (while ZK 3 is `XulElement` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/impl/XulElement.html#>)).

Version History

Version	Date	Content
5.0.6	December 2010	Client-side actions were introduced since 5.0.6

Useful Java Utilities

In this section we introduce some of the most commonly used Java utility classes.

Executions

org.zkoss.zk.ui.Executions^[1]

getCurrent

Executions.getCurrent()^[2]

Retrieves the current execution (request/response).

createComponents

Executions.createComponents()^[3]

Creates components defined in another zul file and attaches them to the current page.

sendRedirect

Executions.sendRedirect()^[4]

Redirects to another URL. If the parameter is left null, it will redirect to the current page.

Sessions

org.zkoss.zk.ui.Sessions^[5]

getCurrent

Sessions.getCurrent()^[6]

Retrieves the current session.

Clients

org.zkoss.zk.ui.util.Clients^[7]

This class offers a collection of methods which manipulate client side via AU Response.

evalJavaScript

Clients.evalJavaScript()^[8]

This method sends an AU Response to execute the given JavaScript on client side, which is the standard way of calling JavaScript from server side in ZK. For example,

```
Clients.evalJavaScript("zk.log('Hi.');" );
```

To handle javascript errors triggered by evalJavascript, org.zkoss.zk.ui.ScriptErrorListener.class^[9] is provided.

focus

Clients.focus(Component component)^[10]

Clients.focus(String selector)^[11]

Focus the given component or the selector matched component.

scrollIntoView

Clients.scrollIntoView(Component cmp)^[12]

Scrolls the parent of the given component, so the given one becomes visible in the view.

Clients.scrollIntoView(String selector)^[13]

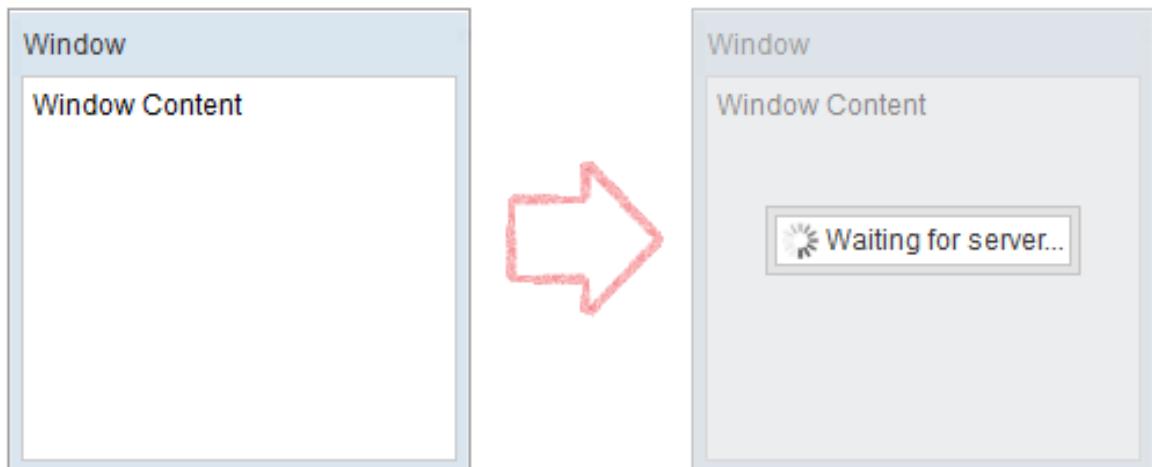
Scrolls the parent of the selector matched component, so the given one becomes visible in the view.

Show Busy Message / Block Users Interaction

Clients.showBusy()^[7] / Clients.clearBusy()^[9]

Display/dismiss a busy message which can cover a component or the whole page. So you can block all interaction to components and let users know the operation is in progress or has finished. For example,

```
Clients.showBusy(window, "Waiting for server...");
```



showNotification

Clients.showNotification()^[14]

It is advised to use Notification class^[15] which was introduced in ZK 9 instead.

Shows a notification box, which is dismissed upon left click (like a Popup). You can either display a global notification (bigger) or a notification specific to another component (smaller with an arrow pointing to the target component).

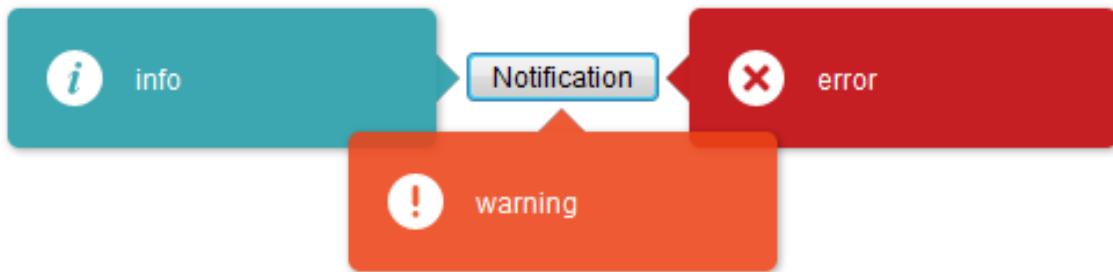
```
Clients.showNotification(msg); // display a global notification box
Clients.showNotification(msg, component); // display a notification box
pointing to a component
```



You can also specify its position, style, and duration (for auto-dismiss):

```
Clients.showNotification(msg, type, component, position, duration);
```

Type determines the style of the notification box.



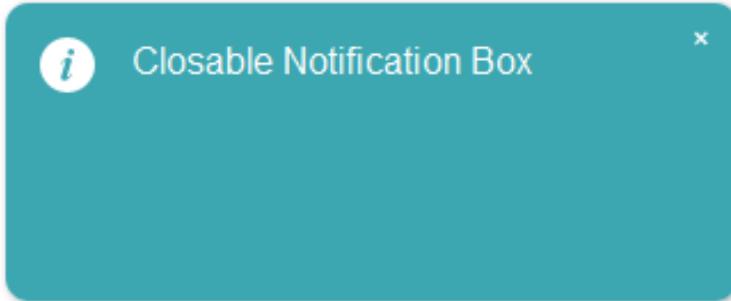
Here are the available positions:



Closable

Notification now supports closable to let users close the notification box manually.

```
// add close icon on the top right corner of notification box  
Clients.showNotification(msg, closable);
```



Multiline

To show a multiline message, just append
 in the message string.

```
Clients.showNotification("msg1 <br/> msg2 <br/>");
```

Notification

org.zkoss.zk.ui.util.Notification^[15]

This class offers a collection of methods showing a notification box, which is dismissed upon left click (like a Popup). You can either display a global notification (bigger) or a smaller notification specific to another component (smaller with an arrow pointing to the target component).

```
Notification.show(msg); // display a global notification box  
Notification.show(msg, component); // display a notification box  
pointing to a component
```

Toast

- Available for ZK:
- CE PE EE

org.zkoss.zkmax.ui.util.Toast^[16]

This class offers a collection of methods showing a toast notification, which is dismissed upon left click (like a Popup). Unlike Notification, Toast is stackable.

Use the force, Harry.



Also possible to do HTML..



```
Toast.show(msg); // display a toast notification
Toast.show(msg, "warning", "top_right"); // display a toast
notification on top-right of the browser viewport
```

You can also specify its position, style, duration (for auto-dismiss) and closable:

```
Toast.show(msg, type, position, duration, closable);
```

Here are the available positions:

	left	center	right
top	top_left	top_center	top_right
middle	middle_left	middle_center	middle_right
bottom	bottom_left	bottom_center	bottom_right

Animation Speed

To specify the duration of opening and closing animation, org.zkoss.zkmax.ui.util.Toast.animationSpeed ^[17] is provided.

```
<library-property>
    <name>org.zkoss.zkmax.ui.util.Toast.animationSpeed</name>
    <value>500</value>
</library-property>
```

Loadingbar

- Available for ZK:
-

org.zkoss.zkmax.ui.util.Loadingbar ^[18]

A Loadingbar behaves like a Progressmeter that provides information on the progress of a task.

You can control a loadingbar with the LoadingbarControl.



```
// create a LoadingbarControl for control the loadingbar
LoadingbarControl loadingbarCtrl = Loadingbar.createLoadingbar("myId");
loadingbarCtrl.start();
loadingbarCtrl.update(20); // update the value to 20
loadingbarCtrl.finish();
```

You can also specify its value(0~100), position(top/bottom) and indeterminate(true/false):

```
loadingbarCtrl.start(20, "top", false);
```

You can turn on/off the indeterminate animation:



```
loadingbarCtrl.update(true); // set loadingbar indeterminate true
```

Animation Speed

To specify the Loadingbar animation speed, org.zkoss.zkmax.ui.util.Loadingbar.animationSpeed^[19] is provided.

```
<library-property>
    <name>org.zkoss.zkmax.ui.util.Loadingbar.animationSpeed</name>
    <value>500</value>
</library-property>
```

Version History

Version	Date	Content
6.0.1	March 2012	Add Clients.showNotification()
6.5.0	July 2012	ZK-1145 ^[20] : Notification supports closable
9.0.0	Sep 2019	ZK-4371 ^[21] : Provide a Toast utility
9.0.0	Sep 2019	ZK-4372 ^[22] : Extract Notification functionalities from Clients
9.0.0	Nov 2019	ZK-4379 ^[23] : Provide a Loadingbar utility
9.5.0	Sep 2020	ZK-4624 ^[24] : Clients API support scrollIntoView and focus by using selector string

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html>
- [2] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#getCurrent\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#getCurrent())
- [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#createComponents%28java.lang.String,%20org.zkoss.zk.ui.Component,%20java.util.Map%29>
- [4] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#sendRedirect%28java.lang.String%29>
- [5] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Sessions.html>
- [6] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Sessions.html#getCurrent\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Sessions.html#getCurrent())
- [7] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html>
- [8] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#evalJavaScript%28java.lang.String%29>
- [9] https://www.zkoss.org/wiki/ZK_Configuration_Reference/zk.xml/The_Library_Properties/org.zkoss.zk.ui.ScriptErrorListener.class
- [10] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#focus\(org.zkoss.zk.ui.Component\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#focus(org.zkoss.zk.ui.Component))
- [11] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#focus\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#focus(java.lang.String))
- [12] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#scrollIntoView\(org.zkoss.zk.ui.Component\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#scrollIntoView(org.zkoss.zk.ui.Component))
- [13] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#scrollIntoView\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#scrollIntoView(java.lang.String))
- [14] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#showNotification\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#showNotification(java.lang.String))
- [15] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Notification.html>
- [16] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkmax/ui/util/Toast.html>
- [17] https://www.zkoss.org/wiki/ZK_Configuration_Reference/zk.xml/The_Library_Properties/org.zkoss.zkmax.ui.util.Toast.animationSpeed
- [18] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkmax/ui/util>Loadingbar.html>
- [19] [http://tracker.zkoss.org/browse/ZK-1145](https://www.zkoss.org/wiki/ZK_Configuration_Reference/zk.xml/The_Library_Properties/org.zkoss.zkmax.ui.util>Loadingbar.animationSpeed
[20] <a href=)
- [21] <http://tracker.zkoss.org/browse/ZK-4371>
- [22] <http://tracker.zkoss.org/browse/ZK-4372>
- [23] <http://tracker.zkoss.org/browse/ZK-4379>
- [24] <http://tracker.zkoss.org/browse/ZK-4624>

HTML Tags

Here we discuss how to use HTML tags directly in a ZUML document. There are several ways as described in the following sections, and you could choose one based on your requirements.

What to consider	<html> component	native namespace	XHTML components	JSP
Update Content Dynamically	Yes	No ^[1]	Yes	No ^[2]
Mix with ZUL components	No	Yes	Yes	Yes/No ^[3]
Memory Footprint	Small	Small	Large	Small
Support EL	Yes	Yes	Yes	Yes
Support Data Binding	Yes	No	Yes	No

In addition, you could use iframe to embed a complete HTML document which might be from a different website with different technology. Or, use include to include an HTML fragment.

-
- [1] We cannot update content dynamically at the server. However, we could modify the DOM tree directly at the client. Please refer to the Client-side UI Composing section.
 - [2] Technically you could modify the browser's DOM tree dynamically at the client.
 - [3] You could mix HTML tags with ZK components, if ZK JSP Tags (<http://www.zkoss.org/product/zkjsp.dsp>) is used. Otherwise, you could only have a JSP page to include other ZUL pages, or vice versa.

The html Component

Overview

One of the simplest ways is to use a XUL component called `html` to embed whatever HTML tags you want. The `html` component works like an HTML SPAN tag enclosing the HTML fragment. For example,

```
<window border="normal" title="Html Demo">
  <html><! [CDATA[
    <h4>Hi, ${parent.title}</h4>
    <p>It is the content of the html component.</p>
  ]]></html>
</window>
```

As shown above, we enclose them with `<! [CDATA[...]>` to prevent ZK Loader from interpreting the HTML tags embedded in the `html` element. In other words, they are not the child component. Rather, they are stored in the `content` property (by use of `Html.setContent(java.lang.String)`^{[1][2]}). In other words, `<h4>...</p>` will become the content of the `html` element.

Also notice that EL expressions are allowed.

-
- [1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Html.html#setContent\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Html.html#setContent(java.lang.String))
 - [2] For more information please refer to [ZUML Reference](#).

What Are Generated

The `html` component will eventually generate the HTML SPAN tag to enclose the content when attached to the browser's DOM tree. In other words, it generates the following HTML tags when attached to the browser's DOM tree.

```
<span id="z_4a_3">
  <h4>Hi, Html Demo</h4>
  <p>It is the content of the html component.</p>
</span>
```

It's a Component

The `html` component is no different to other XUL components. For example, you specify the CSS style and change its content dynamically.

```
<zk>
  <html id="h" style="border: 1px solid blue; background: yellow">
    <! [CDATA[
      <ul>
        <li>Native browser content</li>
    ]]>
```

```
</ul>
]]>
</html>
<button label="change" onClick="h.setContent('Hi, Update')"/>
</zk>
```

You can change its content dynamically.

```
htm.setContent("<ul><li>New content</li></ul>");
```

Limitation

Since SPAN is used to enclose the embedded HTML tags, the following code snippet is incorrect.

```
<zk>
<html><! [CDATA[
    <ul>
        <li> <!-- incorrect since <ul><li> is inside <span> -->
    ]]>
</html>

<textbox />

<html><! [CDATA[
    </li>
    </ul>
]]>
</html>
</zk>
```

If you need to generate the embedded HTML tags directly without the enclosing SPAN tag, you can use the xhtml component set or the native namespace as described in the following section.

The native Namespace

Overview

With the native namespace, an XML element in a ZUML document will be interpreted as a native tag that will be sent to the browser directly, rather than becoming a ZK component.

For example,

```
<n:ul xmlns:n="native">
  <n:li>
    <textbox/>
  </n:li>
  <n:li>
    <textbox/>
  </n:li>
</n:ul>
```

will attach the following HTML tags to the browser's DOM tree:

```
<ul>
  <li>
    <input id="z_a3_2"/>
  </li>
  <li>
    <input id="z_a3_5"/>
  </li>
</ul>
```

where `<input>` is the HTML tag(s) generated by the `textbox` component. Unlike `textbox` in the example above, ZK Loader doesn't really create a component for each of `ul` and `li`.^[1] Rather, they are sent to the client directly. Of course, they must be recognizable by the client. For an HTML browser, they must be valid HTML tags.

[1] ZK actually creates a special component to represent as many XML elements with the native namespace as possible.

Dynamic Update

The XML elements associated with the native namespace will be considered as tags that the client accepts, and they are sent directly to the client to display. They are not ZK components, and they don't have the counterpart (widget) at the client either. The advantage is the better performance in terms of both memory and processing time.

However, the disadvantage that is you cannot access or change them (neither component nor widget) dynamically. For example, the following code snippet is incorrect, since there is no component called `x`.

```
<n:ul id="x" xmlns:n="native"/>
<button label="add" onClick="new Li().setParent(x)"/> <!-- Failed since x is not available at the server -->
```

If you want to change them dynamically, you could:

1. Use client-side code to modify the browser's DOM tree at the client. Notice that, since ZK doesn't create the widget at the client too, you have to manipulate the DOM tree directly.
2. Use the `html` component if you won't mix ZUL with HTML tags.

3. Use XHTML component set as described in the following section.

For example, we can modify the DOM tree with jQuery as follows:

```
<zk xmlns:n="native" xmlns:w="client">
    <n:input id="inp"/>
    <button label="change" w:onClick="jq('#inp')[0].value = 'clicked'" />
</zk>
```

The rule of thumb is to use the native namespace if possible. If you need to change the content dynamically, you might consider the html component first. If still not applicable, use the XHTML component set.

Relation with Other Components

Though no component is associated with the element specified with the native namespace, you still could manipulate its parent, such as invalidate and move. For example, the following works correctly.

```
<window border="normal" title="Redraw">
    <n:ul xmlns:n="native">
        <n:li>ZK is simply best</n:li>
    </n:ul>
    <button label="Redraw" onClick="self.getParent().invalidate()" /><!-- OK to invalidate a component -->
</window>
```

As shown, it is OK to invalidate a component even if it has some native tags.

Also notice that, though the native HTML tags will be generated for the native namespace, ZK Loader actually creates a component to represent as many as these native HTML tags. Thus, if you invoke Component.getPreviousSibling() ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#%23getPreviousSibling\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#%23getPreviousSibling())) of the button above, it will return this component. However, don't access it since the real class/implementation of the component depends on the version you use and might be changed in the future.

In Pure Java

You could use the native namespace in Java too. For example, you could create a native table directly as follows.

```
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.HtmlNativeComponent;
import org.zkoss.zul.Datebox;

public class TableCreator {
    public void createTable(Component parent) {
        HtmlNativeComponent n =
            new HtmlNativeComponent("table", "<tr><td>When:</td><td>", "</td></tr>");
        n.setDynamicProperty("border", "1");
        n.setDynamicProperty("width", "100%");
        n.appendChild(new Datebox());
        parent.appendChild(n);
    }
}
```

As shown, the first argument of `java.lang.String, java.lang.String)` `HtmlNativeComponent.HtmlNativeComponent(java.lang.String, java.lang.String, java.lang.String)` ([\(http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlNativeComponent.html#HtmlNativeComponent\(java.lang.String,%20java.lang.String,%20java.lang.String\)\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlNativeComponent.html#HtmlNativeComponent(java.lang.String,%20java.lang.String,%20java.lang.String))) is the name of tag. The second argument specifies the content that will be generated right before the children, if any. The third specifies the content after the children, if any. In addition, the `setDynamicProperty` method could be used to set the attributes of the tag.

In summary, the above example is equivalent to the following ZUML document:

```
<table border="1" width="100%" xmlns="native" xmlns:u="zul">
    <tr>
        <td>When:</td>
        <td><u:datebox/></td>
    </tr>
</table>
```

Output Tags with Another Namespace

If the HTML tag you want to output requires an XML namespace (such as XAML), you can use the following format to specify the URI of the XML namespace you want to output:

```
<element xmlns="native:URI-of-another-namespace">
```

For example, if you want to output the XAML tags directly to the client, you can specify XAML's XML namespace as follows.

```
<div>
    <Canvas xmlns="native:http://schemas.microsoft.com/client/2007">
        <TextBlock>Hello World!</TextBlock>
    </Canvas>
</div>
```

Then, the result DOM structure will be similar to the following^[1]:

```
<div id="zk_uuid">
    <canvas xmlns="http://schemas.microsoft.com/client/2007">
        <textblock>Hello World!</textblock>
    </canvas>
</div>
```

[1] The real DOM structure of a component (div in this example) depends on its implementation. Here is only a simplified version.

The XHTML Component Set

Overview

Like ZUL, the XHTML component set is a collection of components. Unlike ZUL, which is designed to have rich features, each XHTML component represents an HTML tag. For example, the following XML element will cause ZK Loader to create a component called UI^[1].

```
<h:ul xmlns:h="xhtml">
```

XHTML component supports HTML5 tag attributes, and these attributes could be accessed by MVVM. About MVVM, please refer to the MVVM document^[2].

Dynamic Update

Because Components are instantiated for XML elements specified with the XHTML namespace, you could update its content dynamically on the server. For example, we could allow users to click a button to add a column as shown below.



```
mix HTML demo
column 1 AA ▾

<>window title="mix HTML demo" xmlns:h="xhtml">
  <h:table border="1">
    <h:tr id="row1">
      <h:td>column 1</h:td>
      <h:td>
        <listbox id="list" mold="select">
          <listitem label="AA"/>
          <listitem label="BB"/>
        </listbox>
      </h:td>
    </h:tr>
  </h:table>
  <button label="add" onClick="row1.appendChild(new org.zkoss.zhtml.Td())"/>
</window>
```

On the other hand, the native namespace will cause *native* HTML tags to be generated. It means you can not modify the content dynamically on the server. Notice that you still can handle them dynamically at the client.

However, when an XHTML component is used, a component running on the server has to be maintained. Thus, you should use the XHTML component set only if there is no better way for doing it.

For example, we could rewrite the previous sample with the native namespace and some client-side code as follows.

```
<>window title="mix HTML demo" xmlns:n="native">
  <n:table border="1">
    <n:tr id="row1">
      <n:td>column 1</n:td>
      <n:td>
```

```

<listbox id="list" mold="select">
    <listitem label="AA"/>
    <listitem label="BB"/>
</listbox>
</n:td>
</n:tr>
</n:table>
<button label="add" w:onClick="jq('#row1').append('<td></td>')" xmlns:w="client"/>
</window>

```

ID and UUID

Unlike other components, if you assign ID to an XHTML component, its UUID (Component.getUuid() [3]) is changed accordingly. It means you cannot have two XHTML components with the same ID, no matter if they are in different ID spaces.

Filename Extension

As described in ZUML, the XHTML component set is associated with zhtml, xhtml, html and htm. It means you could name a ZUML page as foo.zhtml if you map *.zhtml to ZK Loader. However, when this kind of file is interpreted, ZK Loader assumes it will have its own HTML, HEAD, BODY tags. On the other hand, these tags are generated automatically if the filename extension is zul.

For example, suppose we have a file called foo.zhtml, then the content might look as follows.

```

<?link rel="shortcut icon" href="/favicon.ico" type="image/x-icon"?>
<html xmlns:zk="zk" xmlns:z="zul">
    <head>
        <title>ZHTML Demo</title>
        <zhead/><!-- a special tag to indicate where to generate ZK CSS and JS files -->
    </head>
    <body style="height:auto">
        <h1>ZHTML Demo</h1>
        <ul id="ul">
            <li>The first item.</li>
            <li id="li2" zk:onClick='self.setId("li2".equals(self.getId()) ? "" :"li2")'>Click me to change Id.</li>
        </ul>
    </body>
</html>

```

where

1. Since the extension is zhtml, the default namespace is XHTML. Thus, we have to specify the zk and zul namespace explicitly.
 - Notice that we have to specify the zk namespace too, because XHTML will cause ZK Loader to consider any unrecognized element as native HTML tag.
2. We have to specify HTML, HEAD and BODY to make it a valid HTML document.
3. We could specify zkhead (line 5) to indicate where to generate ZK CSS and JavaScript files. It is optional. If not specified, ZK will try to identify the proper location for ZK CSS and JavaScript files. Specify it if you want some CSS or JavaScript files to be evaluated *before* or *after* ZK's default ones.

4. By default, BODY's CSS is `width:100%;height:100%`. However, it is appropriate for Web-look page^[4]
For Web-look, we could specify `height:auto` to reset it back to the browser's default.
-

[1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zhtml/Ul.html#>
[2] http://books.zkoss.org/zk-mvvm-book/8.0/introduction_of_mvvm.html
[3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#getUuid\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#getUuid())
[4] `height:100%` is more for desktop-application-look, such as using with Ul (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zhtml/Ul.html#>).

Long Operations

Events for the same desktop are processed sequentially. It simplifies the GUI programming and component development. However, it means an event handler that spends a lot of time to execute will block any following handlers. Worse of all, the user, due to the limitation of HTTP, got no hint but the small busy dialog shown at the left-top corner on the browser.

There are basically two approaches:

1. Handle everything in an event thread and have the user wait but show a more visible message to notify them
2. Handle the long operation in an independent thread, such that the user can access other functions

The first approach could be done with a technique called *echo events* as described in the Use Echo Events section.

The second approach can be done in several ways, such as starting a working thread to do the long operation and then using a timer to check if the data is ready and show to the client. However, there is a simple approach: use an event queue to run an asynchronous listener as described in the Use Event Queues section.

In addition to the above approaches, there is a special mechanism called piggyback, which could be used to piggy back UI updates without extra network traffic.

Use Echo Events

Event echoing is useful for implementing a long operation.

As described in the previous section, HTTP is a request-and-response protocol, so the user won't see any feedback until the request has been served and responded. Thus, if the processing of a request takes too long to execute, the user has no idea if the request is being processed, or they didn't, say, click the button successfully. The user usually tends to click again to ensure it is really clicked, but it only causes the server to respond much slower.

The better approach is to send back some busy messages to let the user know what happens during processing the long operation. It can be done easily with event echoing. If you prefer to allow the user to keep accessing other functions, please refer to the Use Event Queues section, which is powerful but more sophisticated to implement.

Event echoing for a long operation basically takes three steps

1. Invoke Clients.showBusy(java.lang.String)^[7] to show a busy message and block the user from accessing any function
 - Of course, you could have any effect you like, such as showing a modal window.
Clients.showBusy(java.lang.String)^[7] is yet a built-in approach for showing the busy message.
2. Invoke org.zkoss.zk.ui.Component, java.lang.Object) Events.echoEvent(java.lang.String, org.zkoss.zk.ui.Component, java.lang.Object)^[1] to echo an event
3. Listen to the event being echoed and do the long operation in the listener
 - At the end of the event listener, remember to remove the busy message, and update the UI if necessary

For example, assume the long operation is called doLongOperation, then:

```
<window id="w" width="200px" title="Test echoEvent" border="normal">
  <attribute name="onLater">
    doLongOperation(); //take long to execute
    Clients.clearBusy(); //remove the busy message
  </attribute>

  <button label="Echo Event">
    <attribute name="onClick">
      Clients.showBusy("Execute..."); //show a busy message to user
      Events.echoEvent("onLater", w, null); //echo an event back
    </attribute>
  </button>
</window>
```

Better Feedback with Button's autodisable

With event echoing, it might still take hundreds of milliseconds to have the busy message, especially with the slow connection. The feedback to user can be further improved by the use of Button.setAutodisable(java.lang.String)^[2]. For example,

```
<button label="Echo Event" autodisable="self">
  ...
```

Then, the button will be disabled automatically when it is pressed, and enabled automatically when the request has been served.

References

- [1] [`http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Events.html#echoEvent\(java.lang.String\)`](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Events.html#echoEvent(java.lang.String))
- [2] [`http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Button.html#setAutodisable\(java.lang.String\)`](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Button.html#setAutodisable(java.lang.String))

Use Event Queues

The event queue provides a simple way to execute a so-called asynchronous event listener in parallel to other event listeners. Thus, it won't block the user from accessing other functions even if the asynchronous event listener takes a lot of time to execute.

The event queue starts a working thread to invoke the asynchronous event listener, though it is transparent to the caller. Thus, it cannot be used in the environment that does not allow the use of working threads, such as Google App Engine^[1].

In addition, it will start a server push automatically to send the UI updates back when it is ready. If you prefer to use the client polling or a particular implementation, you could start it manually by use of DesktopCtrl.enableServerPush(org.zkoss.zk.ui.sys.ServerPush)^[2], such as:

```
((DesktopCtrl)desktop).enableServerPush(  
    new org.zkoss.zk.ui.impl.PollingServerPush(2000,5000,-1));
```

Example

We provide two implementations to illustrate how to use the event queue's asynchronous listener for executing a long operation. The first approach is more generic that you can modify it to use in more diverse situations. On the other hand, the second approach is much simpler. If you don't have time, you could skip the first approach and study the second approach.

A Generic Approach

A typical use case is to subscribe an asynchronous event listener for doing the long operation, and to subscribe a synchronous event listener to update the user interface. Then, when starting a long operation, an event is posted to the asynchronous event listener for processing. Since the invocation is asynchronous, the user can still interact with ZK smoothly. At the end of the invocation of the asynchronous event listener, it publishes an event to the synchronous event listener to update the result of the long operation back to the browser.

For example,

```
<window title="test of long operation" border="normal">  
    <html><! [CDATA[  
        <ul>  
            <li>Click the button it will start a long operation.</li>  
            <li>With this implementation, you can press the button again even if  
                the long operation is still being processed</li>  
        </ul>  
    ]]></html>  
    <zscript>  
        void print(String msg) {  
            new Label(msg).setParent(inf);  
        }  
    </zscript>
```

```
</zscript>
<button label="async long op">
    <attribute name="onClick"><! [CDATA[
if (EventQueues.exists("longop")) {
    print("It is busy. Please wait");
    return; //busy
}

EventQueue eq = EventQueues.lookup("longop"); //create a queue
String result;

//subscribe async listener to handle long operation
eq.subscribe(new EventListener() {
    public void onEvent(Event evt) {
        if ("doLongOp".equals(evt.getName())) {
            org.zkoss.lang.Threads.sleep(3000); //simulate a long
operation
            result = "success"; //store the result
            eq.publish(new Event("endLongOp")); //notify it is done
        }
    }
}, true); //asynchronous

//subscribe a normal listener to show the resul to the browser
eq.subscribe(new EventListener() {
    public void onEvent(Event evt) {
        if ("endLongOp".equals(evt.getName())) {
            print(result); //show the result to the browser
            EventQueues.remove("longop");
        }
    }
}); //synchronous

print("Wait for 3 seconds");
eq.publish(new Event("doLongOp")); //kick off the long operation
]]></attribute>
</button>
<vbox id="inf"/>
</window>
```

An asynchronous event listener is *not* allowed to access the desktop, but it is allowed to invoke `EventQueue.publish(org.zkoss.zk.ui.event.Event)`^[3] to publish an event.

A Simpler Approach

While subscribing the asynchronous and synchronous event listeners separately is generic, as illustrated above, the event queue provides a simple method to allow you to register them in one invocation: `EventQueue.subscribe(org.zkoss.zk.ui.event.EventListener, org.zkoss.zk.ui.event.EventListener)` [4]. In addition, you don't need to publish an event at the end of the asynchronous event listener -- the synchronous event listener is invoked automatically.

```
<window title="test of long operation" border="normal">
    <zscript>
        void print(String msg) {
            new Label(msg).setParent(inf);
        }
    </zscript>
    <button label="async long op">
        <attribute name="onClick"><![CDATA[
            if (EventQueues.exists("longop")) {
                print("It is busy. Please wait");
                return; //busy
            }

            EventQueue eq = EventQueues.lookup("longop"); //create a queue
            String result;

            //subscribe async listener to handle long operation
            eq.subscribe(new EventListener() {
                public void onEvent(Event evt) { //asynchronous
                    org.zkoss.lang.Threads.sleep(3000); //simulate a long operation
                    result = "success"; //store the result
                }
            }, new EventListener() { //callback
                public void onEvent(Event evt) {
                    print(result); //show the result to the browser
                    EventQueues.remove("longop");
                }
            });
        ]]></attribute>
    </button>
    <vbox id="inf"/>
</window>
```

Better Feedback with Button's autodisable

In the above example, we displayed a message if the button was pressed twice. If you prefer to simply disable the button, you could use `Button.setAutodisable(java.lang.String)`^[2]. For example,

```
<button label="async long op" autodisable="+self">
...

```

Then, the button will be disabled automatically when it is pressed. Notice that we prefix `self` with +, and it means you have to enable it manually (once it is OK to run again).

```
if (ready)
    button.setDisabled(false); //enable it when ready
```

References

- [1] <http://code.google.com/appengine/>
- [2] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/DesktopCtrl.html#enableServerPush\(org.zkoss.zk.ui.sys.ServerPush\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/DesktopCtrl.html#enableServerPush(org.zkoss.zk.ui.sys.ServerPush))
- [3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/EventQueue.html#publish\(org.zkoss.zk.ui.event.Event\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/EventQueue.html#publish(org.zkoss.zk.ui.event.Event))
- [4] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/EventQueue.html#subscribe\(org.zkoss.zk.ui.event.EventListener,](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/EventQueue.html#subscribe(org.zkoss.zk.ui.event.EventListener,)

Use Piggyback

Sometimes there is no hurry to update the result to a client. Rather, the UI update could be sent back when the user, say, clicks a button or triggers some request to the server. This technique is called *piggyback*.

In piggyback, all you need to do is to register an event listener for the `onPiggyback` event to one of the **root components**. Then, the listener will be invoked each time ZK Update Engine has processed an AU request.

For example, suppose we have a long operation that is processed in a working thread, then:

```
<window id="main" title="Working Thread" onPiggyback="checkResult()">
<zscript>
    List result = Collections.synchronizedList(new LinkedList());
    void checkResult() {
        while (!result.isEmpty())
            main.appendChild(result.remove(0));
    }
</zscript>
<timer id="timer" />
<button label="Start Working Thread">
    <attribute name="onClick">
        timer.start();
        new test.WorkingThread(desktop, result).start();
    </attribute>
</button>
</window>
```

The advantage of the piggyback is no extra traffic between the client and the server. However, the user sees no updates if they don't have any activity, such as clicking. Whether it is proper is really up to the application requirements.

Note: A deferrable event won't be sent to the client immediately, so the `onPiggyback` event is triggered only if a non-deferrable event is fired. For more information, please refer to the Deferrable Event Listeners section.

Communication

Here we discuss how to communicate among pages, desktops and Web applications.

Inter-Page Communication

Communicating among pages in the same desktop is straightforward. First, you can use attributes to share data. Second, you can use events to notify each other.

Identify a Page

To communicate among pages, we have to assign an identifier to the target page. In ZUML, it is done by the use of the `page` directive:

```
<?page id=" foo"?>
<window id="main"/>
```

Then we could retrieve it by use of `Desktop.getPage(java.lang.String)`^[1] or by use of a utility class called `Path`^[8]. For example, the following statements could access the `main` window above:

```
comp.getDesktop().getPage("foo").getFellow("main");
Path.getComponent("//foo/main");
```

As shown, `Path.getComponent(java.lang.String)`^[2] considers an ID starting with double slashes as a page's ID.

Use Attributes

Each component, page, desktop, session and Web application has an independent map of attributes. It is a good place to share data among components, pages, desktops and even sessions.

In Java, you could use `"setAttribute()", "removeAttribute()"` and `"getAttribute()` of Component^[1], Page^[8] and so on to share data. Another way is using the scope argument to identify which scope you want to access. (In the following example, assuming `comp` is a component.)

```
comp.setAttribute("some", "anyObject");
comp.getAttribute("some", comp.DESKTOP_SCOPE);
comp.getDesktop().getAttribute("some"); //is equivalent to previous
line
```

In zscript and EL expressions, you could use the implicit objects: `componentScope`, `pageScope`, `desktopScope`, `sessionScope`, `requestScope` and `applicationScope`.

```
<window>
<zscript><! [CDATA[
    desktop.setAttribute("some", "anyObject");
    desktopScope.get("some");
```

```
]]></zscript>
1:${desktopScope["some"]}
</window>
```

Post and Send Events

You could communicate among pages in the same desktop. The way to communicate is to use the `Events.postEvent(org.zkoss.zk.ui.event.Event)`^[3] or `Events.sendEvent(org.zkoss.zk.ui.event.Event)`^[4] to notify a component in the target page.

For example,

```
Events.postEvent(new Event("SomethingHappens",
    comp.getDesktop().getPage("foo").getFellow("main"));
```

You can also pass the data with the event object. The third parameter in `Events.postEvent(org.zkoss.zk.ui.event.Event)`^[3] will be put into `Event.getData()`^[5]. You could pass the data you want with it.

```
Events.postEvent("onTest", target, "this will be send");
```

References

- [1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Desktop.html#getPage\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Desktop.html#getPage(java.lang.String))
- [2] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Path.html#getComponent\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Path.html#getComponent(java.lang.String))
- [3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Events.html#postEvent\(org.zkoss.zk.ui.event.Event\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Events.html#postEvent(org.zkoss.zk.ui.event.Event))
- [4] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Events.html#sendEvent\(org.zkoss.zk.ui.event.Event\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Events.html#sendEvent(org.zkoss.zk.ui.event.Event))
- [5] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Event.html#getData\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Event.html#getData())

Inter-Desktop Communication

Unlike pages, you cannot access two desktops at the same time. You cannot send or post an event from one desktop to another directly either. Rather, we have to use an event queue with a proper scope, such as group, session or application -- depending on where the other desktop is located.

Desktops in the Same Browser Window

In most cases, each browser window has at most one desktop. However, it is still possible to have multiple desktops in one browser window:

- Use HTML IFRAME or FRAMESET to integrate multiple ZUML pages
- Use a portal server to integrate multiple ZK portlets
- Assemble multiple ZUML pages at the client, such as the templating technology described in this section

In this case, you could communicate among desktops by the use of an event queue with the group scope (`EventQueues.GROUP`^[1]).

```
EventQueue que = EventQueues.lookup("groupTest", EventQueues.GROUP,
true);
que.subscribe(new EventListener() {
    public void onEvent(Event evt) {
        //receive event from this event queue (within the same
        group of desktops)
    }
});
```

Notice that the desktop-scoped event queue does not require Server Push, so there is no performance impact at all.

Here is a dumb example: chat among iframes.

```
<!-- main -->
<window title="main" border="normal" onOK="publish()">
<zscript>
EventQueue que = EventQueues.lookup("groupTest", "group", true);
que.subscribe(new EventListener() {
    public void onEvent(Event evt) {
        o.setValue(o.getValue() + evt.getData() + "\n");
    }
});
void publish() {
    String text = i.getValue();
    if (text.length() > 0) {
        i.setValue("");
        que.publish(new Event("onGroupTest", null, text));
    }
}
</zscript>
Please enter:
<textbox id="i" onChange="publish()"/>
<textbox id="o" rows="6"/>
```

```

<separator/>
<iframe src="includee.zul" height="500px" width="30%"/>
<iframe src="includee.zul" height="500px" width="30%"/>
<iframe src="includee.zul" height="500px" width="30%"/>
</window>

```

And, this is the ZUML page being referenced (by iframe).

```

<!-- includee.zul -->
<window title="frame2" border="normal" onOK="publish()" >
    <zscript>
        EventQueue que = EventQueues.lookup("groupTest", "group",
true);
        que.subscribe(new EventListener() {
            public void onEvent(Event evt) {
                o.setValue(o.getValue() + evt.getData() +
"\n");
            }
        });
        void publish() {
            String text = i.getValue();
            if (text.length() > 0) {
                i.setValue("");
                que.publish(new Event("onGroupTest", null,
text));
            }
        }
    </zscript>
    <textbox id="i" onChange="publish()" />
    <textbox id="o" rows="6" />
</window>

```

Desktop in Different Sessions

Similarly, we could use an event queue to communicate among desktops belonging to different sessions. The only difference is to specify EventQueues.APPLICATION^[2] as the scope.

```

EventQueue que = EventQueues.lookup("groupTest",
EventQueues.APPLICATION, true);

```

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/EventQueues.html#GROUP>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/EventQueues.html#APPLICATION>

Inter-Application Communication

An EAR file or the installation of Web server could have multiple WAR files. Each of them is a Web application. There are no standard way to communicate between two Web applications. However, there are a few ways to work around it.

Include Another Application's Resource with ZK URL Prefix (~APP_CONTEXT/)

ZK supports a way to reference the resource from another web application on the same application server. For example, assume you want to include a resource, say /foreign.zul, from another Web application, say app2. Then, you could do as follows.

```
<include src="~app2/foreign.zul"/>
```

Similarly, you could reference resources from another Web application.

```
<style src="~app2/foo.css"/> <!-- assume foo.css is in the context called app2 -->
<image src="~/foo.png"/> <!-- assume foo.png is in the root context -->
```

Note: Whether you can access a resource located in another Web application depends on the configuration of the Web server. For example, you have to specify `crossContext="true"` in `conf/context.xml`, if you are using Tomcat.

Limitation

Cross-context access is not always allowed in a container, e.g. Tomcat, you need to enable `crossContext` ^[1] first before including another context resources.

Use Cookie

Cookie ^[2] is another way to communicate among Web applications. It can be done by setting the path to "/", such that every Web application in the same host will see it.

```
HttpServletResponse response =
(HttpServletResponse) Executions.getCurrent().getNativeResponse();
Cookie userCookie = new Cookie("user", "foo");
userCookie.setPath("/");
response.addCookie(userCookie);
```

Web Resources from Classpath

Though it is not necessary for inter-application communication, you could, with ZK, reference a resource that is locatable by the classpath. The advantage is that you could embed Web resources in a JAR file, which simplifies the deployment. Please read [Include_a_Page#Classpath_Web_Resource_Path](#)^[3].

References

[1] <https://tomcat.apache.org/tomcat-9.0-doc/config/context.html>

[2] http://en.wikipedia.org/wiki/HTTP_cookie

[3] https://www.zkoss.org/wiki/ZK_Developer%27s_Reference/UI_Composing/ZUML/Include_a_Page#Classpath_Web_Resource_Path

Templating

Templating is a technique that allows developers to define UI fragments, and how to assemble them into a complete UI at runtime. With ZK, it can be done by the use of annotations and composers (or initiators, utilInitiator^[1]).

In general, templating can be done by specifying the name of a fragment as annotations in a ZUML document that shall represent a complete UI, and a composer that is capable to parse annotations and replace them with the fragment. For example,

```
<div apply="foo.MyTemplateManager"><!-- your template manager -->
    <include src="@header()"/><!-- you could use any component as long as your manager knows how to handle it -->
    <include src="@content()"/>
    <include src="@footer()"/>
</div>
```

Here is a list of the implementations that ZK supports by default. You could implement your own, if it does not fulfill your requirement. If the templating is stateful and dynamical, you might consider ZK Spring^[2] for using Spring Web Flow instead.

References

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

[2] <http://www.zkoss.org/product/zkspring.dsp>

Composition

Composition^[1] is one of the built-in templating implementations. The concept is simple:

1. Define a template (a ZUML document representing a complete UI)
2. Define a ZUML document that contains a collection of fragments that a template might reference

Notice that the user shall visit the ZUML document with a collection of fragments rather than the template document.

The advantage of Composition^[1] is that you don't need additional configuration file.

Note: the composition doesn't support mixing up ZUML and ZHTML language, that is, if you define a ZHTML template as the HTML content that contains *Html* and *Body* tags, you cannot use that template in a ZUML page.

Defines a Template

A template document is a ZUML document that defines how to assemble the fragments. For example,

```
<!-- /WEB-INF/layout/template.zul -->
<vbox>
  <hbox self="@insert(content)" />
  <hbox self="@insert(detail)" />
</vbox>
```

As shown, the anchor (i.e., the component that a fragment will insert as children) is defined by specifying an annotation as `@insert(name)`. Then, when Composition^[1] is applied to a ZUML document with a collection of fragments, the matched fragment will become the child of the annotated component (such as `hbox` in the above example).

Define Fragments

To apply a template to a ZUML document that a user visits, you have to define a collection of fragments that a template might use, and then specify Composition^[1] as one of the initiators of the document:

```
<!-- foo/index.zul -->
<?init class="org.zkoss.zk.ui.util.Composition"
arg0="/WEB-INF/layout/template.zul"?>
<zk>
  <window self="@define(content)" title="window1" width="100px"/>
  <window self="@define(content)" title="window2" width="200px"/>
  <grid self="@define(detail)" width="300px" height="100px"/>
</zk>
```

As shown, a fragment is defined by specifying an annotation as `self="@define(name)"`. Furthermore, the template is specified in the init directive.

Then, when the user visits this page (`foo/index.zul` in the above example), Composition^[1] will:

1. Load the template, and render it as the root components of this page(`foo/index.zul`)
2. Move the fragments specified in this page to become the children of the anchor component with the same annotation name

Thus, here is the result

```
<vbox>
  <hbox>
    <window title="window1" width="100px"/>
    <window title="window2" width="200px"/>
  </hbox>
  <hbox>
    <grid width="300px" height="100px"/>
  </hbox>
</vbox>
```

Multiple Templates

You could apply multiple templates to a single page too:

```
<?init class="org.zkoss.zk.ui.util.Composition"
arg0="/WEB-INF/layout/template0.zul"
arg1="/WEB-INF/layout/template1.zul"?>
```

The templates specified in `arg0` and `arg1` (etc.) will be loaded and rendered one-by-one.

Grouping Fragments into Separated Files

In a complex templating environment, it might not be appropriate to put fragments in the target page (e.g., `foo/index.zul` in the above example), since you might want to use the same collection of fragments in several target pages. It can be easily by use of the include component as follows.

```
<!-- foo/index.zul -->
<?init class="org.zkoss.zk.ui.util.Composition"
arg0="/WEB-INF/layout/template.zul"?>
<include src="/WEB-INF/layout/fragments.zul"/>
```

Then, you could group fragments into one or multiple individual ZUL documents, such as

```
<!-- /WEB-INF/layout/fragments.zul -->
<zk>
  <window self="@define(content)" title="window1" width="100px"/>
  <window self="@define(content)" title="window2" width="200px"/>
  <grid self="@define(detail)" width="300px" height="100px"/>
</zk>
```

Positioning

If you want to use Composition ^[1] inside any of the containers (like Div, Window, Tabbox), you have to use the include component and set its mode *Defer*:

Note: You have to specify Composition ^[1] as of the initiators of the 'fragments')

```
<!-- foo/index.zul -->
<window title="This is a window" border="normal">
  <include src="/WEB-INF/layout/fragments.zul" mode="defer" />
</window>
```

```
<!-- /WEB-INF/layout/fragments.zul -->
<?init class="org.zkoss.zk.ui.util.Composition" arg0="/WEB-INF/layout/template.zul"?>
<zks>
    <window self="@define(content)" title="window1" width="100px"/>
    <window self="@define(content)" title="window2" width="200px"/>
    <grid self="@define(detail)" width="300px" height="100px"/>
</zks>
```

References

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

Templates

As described in the MVC: Template section, a template is a ZUML fragment that defines how to create components. A template is enclosed with the template element as shown below.

```
<window>
    <template name="foo">
        <textbox/>
        <grid model=${data}>
            <columns/>
            <template name="model"> <!-- nested template -->
                <row>Name: <textbox value=${each.name} /></row>
            </template>
        </grid>
    </template>
    ...

```

Using Template in Application

"Template" is a generic feature and its use is not limited to custom model rendering. Users are able to use "template" in ZK applications too.

Each template is stored as part of a component and can be retrieved by invoking the Component.getTemplate(java.lang.String) [1]. To create the components defined in the template, just invoke the org.zkoss.zk.ui.Component, org.zkoss.xel.VariableResolver, org.zkoss.zk.ui.util.Composer) Template.create(org.zkoss.zk.ui.Component, org.zkoss.zk.ui.Component, org.zkoss.xel.VariableResolver, org.zkoss.zk.ui.util.Composer) [2]. For example,

```
comp.getTemplate("foo").create(comp, null, null, null);
```

The third argument of the create method is a variable resolver (VariableResolver [2]). Depending on the requirement, you could pass any implementation you like. For example, the implementation of a listbox actually utilizes it to return the data being rendered; the code is similar to the following (for easy understanding, the code has been simplified).

For more detailed information about the variable resolver, please refer to ZUML Reference.

```
public class TemplateBasedRenderer implements ListitemRenderer {
    public void render(Listitem item, final Object data, int index) {
```

```

    final Listbox listbox = (Listbox)item.getParent();
    final Component[] items =
listbox.getTemplate("model").create(listbox, item,
        new VariableResolver() {
            public Object resolveVariable(String name) {
                return "each".equals(name) ? data: null;
            }
        }, null);

    final Listitem nli = (Listitem)items[0];
    if (nli.getValue() == null) //template might set it
        nli.setValue(data);
    item.detach();
}
}
}

```

In addition, the template allows users to specify any number of parameters with any name, and these parameters can be retrieved back by the `getParameters` method of the `Template` interface:

```

<template name="foo" var1="value1" var2="${el2}">
...
</template>

```

If the content of a template is located elsewhere as a separate file, to reference it, specify it in the `src` attribute as follows.

```

<template name="foo" src="foo.zul">
...
</template>

```

Children Binding

We suggest using shadow component `<forEach>`^[2] as a replacement of children binding.

ZK Data Binding provides a powerful way called Children Binding to render a template based on the data (such as a list of elements). Moreover, the UI will be updated automatically if the data has been changed. For more information, please refer to the Children Binding section.

Version History

Version	Date	Content
6.0.0	July 2011	The template feature was introduced.

References

- [1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#getTemplate\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#getTemplate(java.lang.String))
- [2] http://books.zkoss.org/zk-mvvm-book/8.0/shadow_elements/iterate_collections.html

XML Output

In addition to generating HTML output to a browser, ZK could be used to generate (static) XML output to any client that recognizes it, such as RSS^[1] and Web Services^[2].

Using ZK to generate XML output is straightforward:

1. Uses the XML component set (<http://www.zkoss.org/2007/xml> and shortcut is `xml`).
2. Maps the file extension to ZK Loader
3. Maps the file extension to the XML component set

The XML component set also provides some special components, such as transformer that supports XSTL. For more information please refer to XML Components.

Use the XML Component Set

The XML component set (aka., the XML language, in ZK terminology) is used to generate XML output. Unlike the ZUL or XHTML component sets, all unknown^[3] tags in a ZUML document are assumed to belong to the native namespace. It means ZK generates them directly to the output without instantiating a ZK component for each of them.

The following is an example that generates the SVG output. It looks very similar to the XML output you want to generate, except you can use zscript, EL expressions, macro components and other ZK features.



```
<?page contentType="image/svg+xml; charset=UTF-8"?>
<svg width="100%" height="100%" version="1.1" xmlns="http://www.w3.org/2000/svg"
xmlns:z="zk">
<z:zscript><![CDATA[
String[] bgnds = {"purple", "blue", "yellow"};
int[] rads = {30, 25, 20};
]]></z:zscript>
<circle style="fill:${each}" z:forEach="${bgnds}"
cx="${50+rads[forEachStatus.index]}"
cy="${20+rads[forEachStatus.index]}"
r="${rads[forEachStatus.index]}"/>
</svg>
```

The generated output will be

```
<svg xmlns="http://www.w3.org/2000/svg" width="100%" height="100%"
version="1.1">
<circle style="fill:purple" cx="80" cy="50" r="30">
</circle>
<circle style="fill:blue" cx="75" cy="45" r="25">
```

```

</circle>
<circle style="fill:yellow" cx="70" cy="40" r="20">
</circle>
</svg>

```

where

- The content type is specified with the page directive. For SVG, it is `image/svg+xml`. The `xml` processing instruction (`<?xml ?>`) and DOCTYPE of the output are also specified in the page directive.
- All tags in this example, such as `svg` and `circle`, are associated with a namespace (`http://www.w3.org/2000/svg`) that is unknown to ZK Loader. Thus, they are assumed to belong to the native namespace. They are output directly rather than instantiating a ZK component for each of them.
- To use `zscript`, `forEach` and other ZK specific features, you have to specify the ZK namespace (`zk`).

[1] <http://www.whatisrss.com/>
[2] http://en.wikipedia.org/wiki/Web_service
[3] By the unknown tag we mean an XML element that is not associated with an XML namespace, or the namespace is unknown.

Maps the File Extension to ZK Loader

To let ZK Loader process the file, you have to associate it with the ZK Loader in `WEB-INF/web.xml`. In this example, we map all files with the `.svg` extension to ZK Loader^[1]:

```

<servlet-mapping>
  <servlet-name>zkLoader</servlet-name>
  <url-pattern>*.svg</url-pattern>
</servlet-mapping>

```

[1] We assume ZK Loader (`zkLoader`) is mapped to `org.zkoss.zk.ui.http.DHtmlLayoutServlet`.

Maps the File Extension to the XML Component Set

Unless the file extension is `.xml`, you have to associate it with the XML component set (aka., the XML language) explicitly in `WEB-INF/zk.xml`. In this example, we map `.svg` to the XML component set:

```

<language-mapping>
  <language-name>xml</language-name>
  <extension>svg</extension>
</language-mapping>

```

where `xml` is the language name of the XML component set. Thus, when ZK Loader parses a file with the `.svg` extension, it knows the default language is the XML component set^[1].

[1] For more information about language identification, please refer to ZUML Reference.

Event Threads

ⓘ Notice: according to Java Servlet Specification that may prohibit the creation of new threads

By default, ZK processes an event in the same Servlet thread that receives the HTTP request. It is the suggested approach because the performance is better, and it is easy to integrate with other frameworks. (Many frameworks store per-request information in the thread-local storage, so we have to copy them from a servlet thread to the Event Processing Thread).

However, it also implies the developer cannot suspend the execution. Otherwise, the end-users won't see any updates from the server. To solve it, ZK provides an alternative approach: processes the event in an independent thread called the event processing thread. Therefore, the developer can suspend and resume the execution at any time, without blocking the Servlet thread from sending back the responses to the browser. To turn it on, you have to specify the following in WEB-INF/zk.xml (ZK Configuration Guide: disable-event-thread , after ZK 5, the event processing thread is disabled by default.)

```
<system-config>
    <disable-event-thread>false</disable-event-thread>
</system-config>
```

In short, it is recommended to disable the event thread. Enable the event thread only if the project does not need to integrate other frameworks (such as Spring), depending on Messagebox [1] and modal windows a lot, and do not have a lot of concurrent users.

Here are the advantages and limitations of using the Servlet thread to process events. In the following sections, we will talk more about the limitations and workarounds when using the Servlet thread.

	Using Servlet Thread	Using Event Processing Thread
Integration	<p>Less integration issues.</p> <p>Many containers assume the HTTP request is handled in the Servlet thread, and many frameworks store per-request information in the thread-local storage.</p>	<p>You may have to implement EventThreadInit and/or EventThreadCleanup to solve the integration issue, such as copying the per-request information from the Servlet thread to the event processing thread.</p> <p>There are several implementations to solve the integration issue, such as HibernateSessionContextListener [1] (they can be found under the org.zkoss.zkplus package [2]).</p>
SuspendResume	<p>No way to suspend the execution of the event listener.</p> <p>For example, you cannot create a modal window.</p>	No limitation at all.
Performance	No extra cost	<p>It executes a bit slower to switch from one thread to another, and it might consume a lot more memory if there are a lot of suspended event processing threads.</p>

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkplus/hibernate/HibernateSessionContextListener.html#>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkplus/package-summary.html>

Modal Windows

Modal Windows with Servlet Thread

When the event is processed in the Servlet thread (default), the execution cannot be suspended. Thus, the modal window behaves the same as the highlighted window (`Window.doHighlighted()`^[1]). At the client side, the visual effect is the same: a semi-transparent mask blocks the end user from access components other than the modal window. However, at the server side, it works just like the overlapped mode – it returns immediately without waiting for user's closing the window.

```
win.doModal(); //returns once the mode is changed; not suspended  
System.out.println("next");
```

The "next" message will be printed to the console before the end user closes the modal window.

Migrate Your Code from Event Thread

With the Event thread, you might write your business logic right after `doModal()`.

```
win.doModal();  
doMyTask(); //your business logic, need to move it for a servlet thread
```

Since now servlet thread doesn't stop at `doModal()`, you need to move your code to another place. You can put it at:

- Window's `onClose` event listener
- add a button in the modal window and call `doMyTask()` in an `onClick` listener and close the modal window.

Modal Windows with Event Thread

If the event thread is enabled, `Window.doModal()`^[2] will suspend the current thread. Thus, the "next" message won't be shown, until the modal window is closed.

When the event thread is suspended, the Servlet thread will be resumed and continue to look another event thread to process other events, if any. Thus, the end user still have the control (such that he can close the modal window if he want).

References

- [1] [`http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Window.html#doHighlighted\(\)`](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Window.html#doHighlighted())
- [2] [`http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Window.html#doModal\(\)`](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Window.html#doModal())

Message Box

Message Boxes with Servlet Thread

When Messagebox.show(java.lang.String) ^[1] is called, it returns immediately after showing the message dialog. Furthermore, it always returns Messagebox.OK. Thus, it is meaningless to show buttons other than the OK button. For example, the `if` clause in the following example is never true.

```
if (Messagebox.show("Delete?", "Prompt", Messagebox.YES | Messagebox.NO,
    Messagebox.QUESTION) == Messagebox.YES) {
    this_never_executes();
}
```

Rather, you have to provide an event listener as follows.

```
Messagebox.show("Delete?", "Prompt", Messagebox.YES | Messagebox.NO,
    Messagebox.QUESTION,
    new EventListener() {
        public void onEvent(Event evt) {
            switch (((Integer)evt.getData()).intValue()) {
                case Messagebox.YES: doYes(); break; //the Yes button is
pressed
                case Messagebox.NO: doNo(); break; //the No button is
pressed
            }
        }
    );
}
```

The event listener you provided is invoked when the user clicks one of the buttons. Then, you can identify which button is clicked by examining the data (Event's `getData`). The data is an integer whose value is the button's identifier, such as `Messagebox.YES`.

Alternatively, you can examine the event name:

```
public void onEvent(Event evt) {
    if ("onYes".equals(evt.getName())) {
        doYes(); //the Yes button is pressed
    } else if ("onNo".equals(evt.getName())) {
        doNo(); //the No button is pressed
    }
}
```

Note: The event name for the OK button is `onOK`, not `onOk`. **Notice:** If you want to make it run under clustering environment, you shall implement `SerializableEventListener` ^[2]. For more information, please refer to ZK Developer's Reference: Clustering.

Message Boxes with Event Thread

If the event thread is enabled, `Messagebox.show(java.lang.String)` ^[1] will suspend the thread until the end user makes the choice. Thus, the following code works correctly.

```
if (Messagebox.show("Delete?", "Prompt", Messagebox.YES | Messagebox.NO,
    Messagebox.QUESTION) == Messagebox.YES) {
    //execute only if the YES button is clicked
}
```

References

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

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

File Upload

File Upload with Servlet Thread

When the event thread is disable (default), it is recommended to use `Button` ^[2], `Toolbarbutton` ^[1] or `MenuItem` ^[2] with `upload="true"` instead. For example,

```
<zk>
    <zscript>
        void upload(UploadEvent event) {
            org.zkoss.util.media.Media media = event.getMedia();
            if (media instanceof org.zkoss.image.Image) {
                org.zkoss.zul.Image image = new
org.zkoss.zul.Image();
                image.setContent( (org.zkoss.image.Image) media);
                image.setParent(pics);
            } else {
                Messagebox.show("Not an image: "+media, "Error",
Messagebox.OK, Messagebox.ERROR);
            }
        }
    </zscript>
    <button label="Upload" upload="true" onUpload="upload(event)" />
    <toolbarbutton label="Upload" upload="true" onUpload="upload(event)" />
    <vbox id="pics" />
</zk>
```

If you prefer to use a dialog (`Fileupload.get()` ^[1]), please take a look at ZK Component Reference: `Fileupload` for more information.

File Upload with Event Thread

If the event thread is disabled, the developer can use Button [2] or Toolbarbutton [1] with upload="true" instead. They behave the same no matter if the event thread is disabled or not.

However, if the event thread is enabled, you can get uploaded Media returned by Fileupload.get() [1] and other overloaded static methods.

```
<zk>
    <button label="Upload">
        <attribute name="onClick">{
            org.zkoss.util.media.Media[] media = Fileupload.get(-1);
            if (media != null) {
                for (int i = 0; i < media.length; i++) {
                    if (media[i] instanceof org.zkoss.image.Image)
{
                        org.zkoss.zul.Image image = new
org.zkoss.zul.Image();
                        image.setContent(media[i]);
                        image.setParent(pics);
                    } else {
                        Messagebox.show("Not an image:
"+media[i], "Error", Messagebox.OK, Messagebox.ERROR);
                        break; //not to show too many errors
                    }
                }
            }
        }</attribute>
    </button>
    <vbox id="pics" />
</zk>
```

As shown, Fileupload.get(int) [2] won't return until an end user uploads the files (and/or closes the dialog).

References

- [1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Fileupload.html#get\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Fileupload.html#get())
- [2] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Fileupload.html#get\(int\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Fileupload.html#get(int))

Theming and Styling

This chapter introduces how you change a component's appearance through different ways including mold, CSS class, or even a theme.

To customize an individual component's appearance, please read [ZK Style Customization Guide](#).

To apply an overall, consistent design on all components, please see [ZK Developer's Reference/Theming and Styling/Creating Custom Themes](#).

Molds

A component could have multiple different visual appearances. Each appearance is called a **mold**. A mold is basically a combination of a DOM structure plus CSS. That is, it is the visual representation of a component. Developers could dynamically change the mold by use of Component.setMold(java.lang.String) ^[1].

All components support at least a mold called `default`, which is the default value. Some components might have support for two or more molds. For example, tabbox supports both `default` and `accordion` molds.

If `tabbox`'s `mold` is not set, it uses the default mold.

```
<tabbox>
    <tabs>
        <tab label="First tab" />
        <tab label="Second tab" />
    </tabs>
    ...
</tabbox>
```

And you could set `tabbox`'s mold to "accordion" to change the look.

```
<tabbox mold="accordion">
    <tabs>
        <tab label="First tab" />
        <tab label="Second tab" />
    </tabs>
    ...
</tabbox>
```

Supported Mold

To know which mold a component supports, please refer to ZK Component Reference.

Custom Mold

To largely change how a component renders in a browser, see ZK_Client-side_Reference/Customization/Custom_Mold.

References

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

CSS Classes and Styles

CSS (Cascading Style Sheets^[1]) is a style sheet language used to describe the presentation of a (HTML) document. It is an important part of ZK to customize component's look and feel. If you are not familiar with CSS, please refer to CSS Tutorial^[2].

There are a set of methods that could be used to set CSS styles for an individual component.

- `HtmlBasedComponent.setStyle(java.lang.String)`^[3] assigns CSS styles directly to a component.
- `HtmlBasedComponent.setSclass(java.lang.String)`^[4] (i.e., sclass) assigns one or multiple CSS style classes to a component.
- `HtmlBasedComponent.setZclass(java.lang.String)`^[5] (i.e., zclass) assigns the main CSS style class to a component. Unlike style and sclass, if zclass is changed, all default CSS styles won't be applied.
- Some components have a so-called content area and they have a separate set of methods to change the CSS style of the content area, such as `Window.setContentStyle(java.lang.String)`^[6] and `Window.setContentSclass(java.lang.String)`^[7].

Notice that the DOM structures of many ZUL components are complicate, and CSS customization might depend on the DOM structure. For more information about how individual component is styled, please refer to ZK Style Guide.

style

Specifying the style is straightforward:

```
<textbox style="color: red; font-style: oblique;" />
```

or, in Java:

```
Textbox tb = new Textbox();
tb.setStyle("color: red; font-style: oblique;");
```

sclass

In addition, you could specify the style class by use of `HtmlBasedComponent.setSclass(java.lang.String)`^[4], such that you could apply the same CSS style to multiple components.

```
<window>
  <style>
    .red {
      color: blue;
      font-style: oblique;
    }
  </style>
  <textbox sclass="red" /> <!-- first textbox -->
  <textbox sclass="red" /> <!-- another textbox -->
</window>
```

You could apply multiple style classes too. As shown below, just separate them with a space.

```
<textbox sclass="red error"/>
```

zclass

Like `sclass`, `zclass` is used to specify the CSS style class. However, `zclass` is the main CSS that each mold of each component has. If it is changed, all default CSS of the given component won't be applied. In other words, you have to provide a full set of CSS rules that a component's mold has.

Rule of thumb: specify `zclass` if you want to customize the look completely. Otherwise, use `sclass` to customize one or a few CSS styles.

For more information, please refer to [ZK Style Guide](#).

content style and sclass

Some container components such as `window`, `groupbox`, `detail` have a content block, you have to use `contentStyle` to set its style.

For example,

```
<window title="below is content" contentStyle="background:yellow">
  Hello, World!
</window>
```

Scollable Window

A typical use of `contentStyle` is to make a window scrollable as follows.

```
<window title="Scroll Example" width="150px" height="100px" contentStyle="overflow:auto" >
This is a long line to spread over several lines, and more content to
display.
Finally, the scrollbar becomes visible.
This is another line.
</window>
```

References

- [1] http://en.wikipedia.org/wiki/Cascading_Style_Sheets
- [2] <http://www.w3schools.com/css/default.asp>
- [3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlBasedComponent.html#setStyle\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlBasedComponent.html#setStyle(java.lang.String))
- [4] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlBasedComponent.html#setSclass\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlBasedComponent.html#setSclass(java.lang.String))
- [5] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlBasedComponent.html#setZclass\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/HtmlBasedComponent.html#setZclass(java.lang.String))
- [6] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Window.html#setContentStyle\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Window.html#setContentStyle(java.lang.String))
- [7] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Window.html#setContentSclass\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Window.html#setContentSclass(java.lang.String))

ZK Official Themes

Overview

This is an overview of the default themes and applicable add-on themes for each ZK version.

	ZK 6.5	ZK 7.0	ZK 8.0	ZK 8.5	ZK 8.6	ZK 10+
Default Theme	Breeze	Breeze	Breeze	Iceblue	Iceblue	Iceblue
Applicable Themes	Sapphire Silvertail	Sapphire Silvertail Atlantic	Sapphire Silvertail Atlantic	Sapphire Silvertail Atlantic ZK Theme Pack	Iceblue Compact Sapphire Silvertail Atlantic ZK Theme Pack Theme Pack Compact	Iceblue Compact ZK Theme Pack Theme Pack Compact

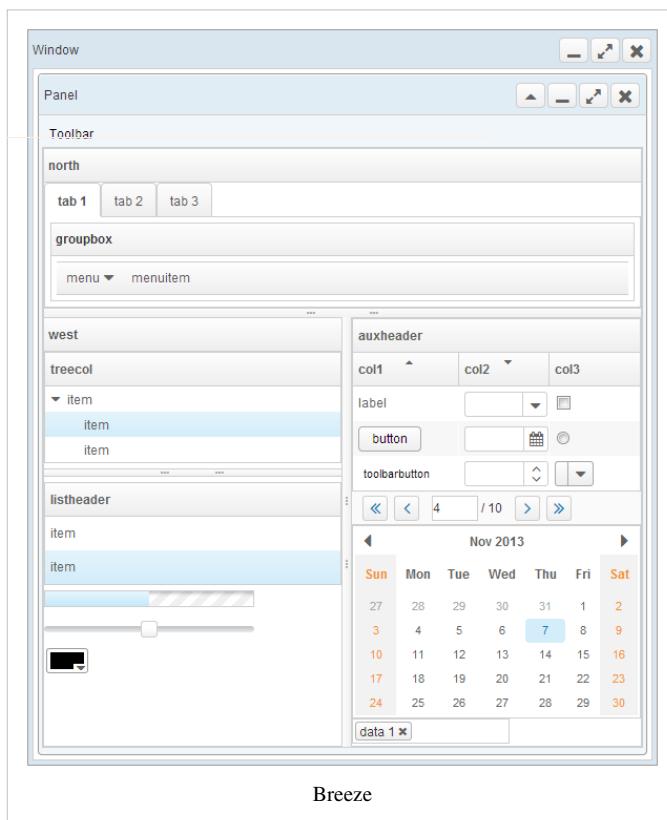
- Theme pack contains 23 themes.
- Theme Pack Compact contains 23 compact themes.

Trendy Design Themes (Deprecated): Breeze, Sapphire, Silvertail

 **Notice:** Deprecated Since 10.0.0, please use ZK Theme Pack ^[1] instead.

Trendy design emphasizes on gradient background, rounded corners and shadow effects. **Breeze** is a greyish based theme that supports desktop and tablet, **sapphire** is a blueish based theme that supports desktop only and **silvertail** is a silverish based theme that also support desktop only.

See below to take a quick view at the look and feel of Breeze, Sapphire and Silvertail, click image to view original size.



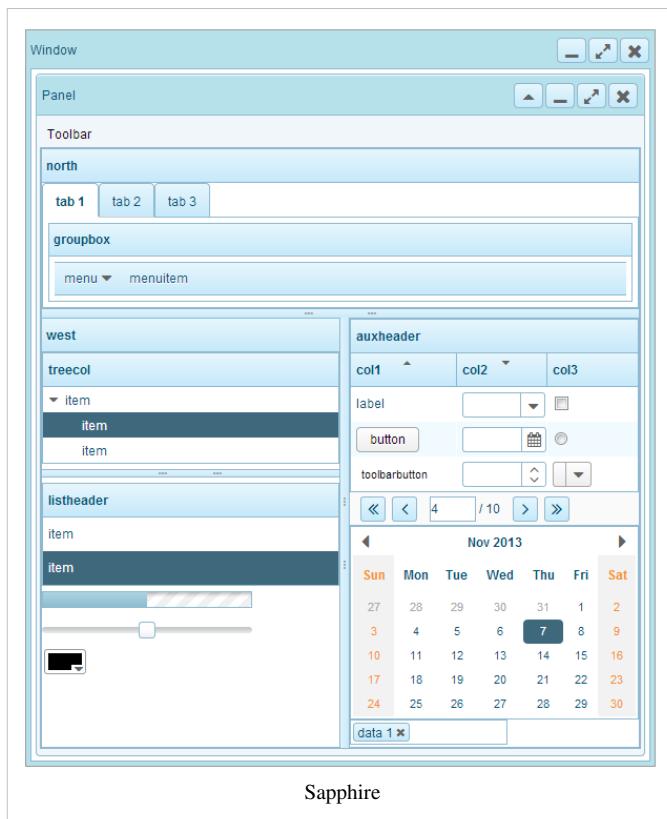
Flat Design Theme (Deprecated): Atlantic

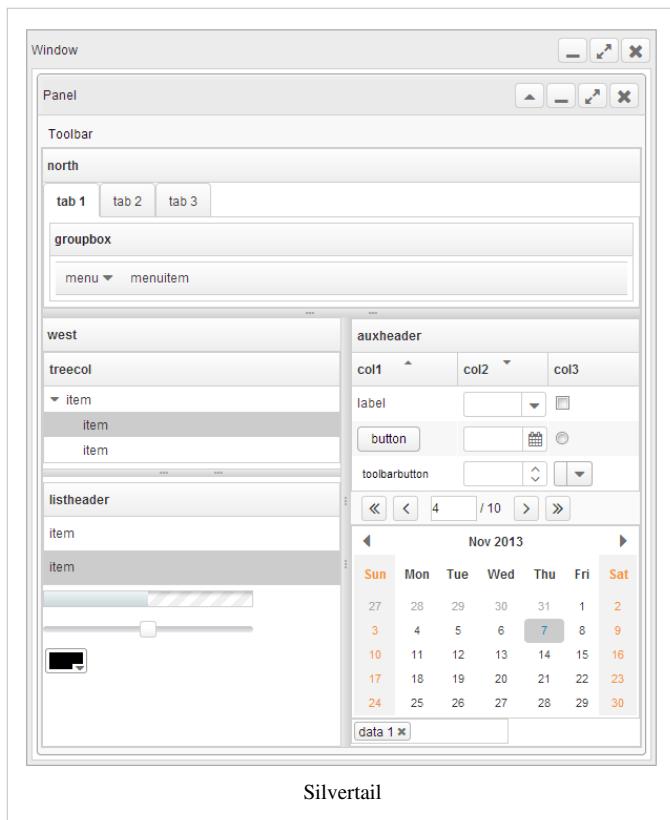
Notice:

please use [ZK Theme Pack \[1\]](#) instead.

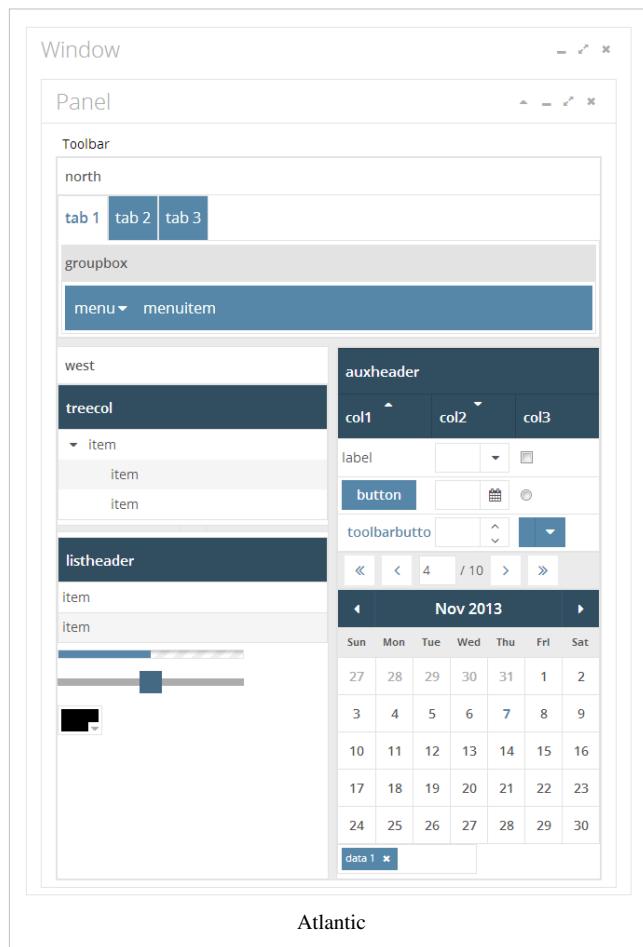
On the contrary, flat design replaces skeuomorphism (gradient background, rounded corner and shadow effect) with simplicity and visual clarity to communicate. It is easy to create and customize a new theme based on this design. **Atlantic** is a blueish based theme that supports both desktop and tablets.

See below to take a quick view at the look and feel of **Atlantic**, click image to view original size.





Silvertail



Atlantic

Not to Import Google Font

Please refer to ZK Configuration Properties/org.zkoss.theme.atlantic.useGoogleFont.disabled

Reference/zk.xml/The

Library

ZK Theme Pack

ZK Theme Pack (Live demo^[2]) contains 23 modern themes, including lite themes, dark themes and mix-match themes. These themes are designed for ZK 8.5 and later versions and are compatible with ZK 8.5's default Iceblue theme.

How to Apply

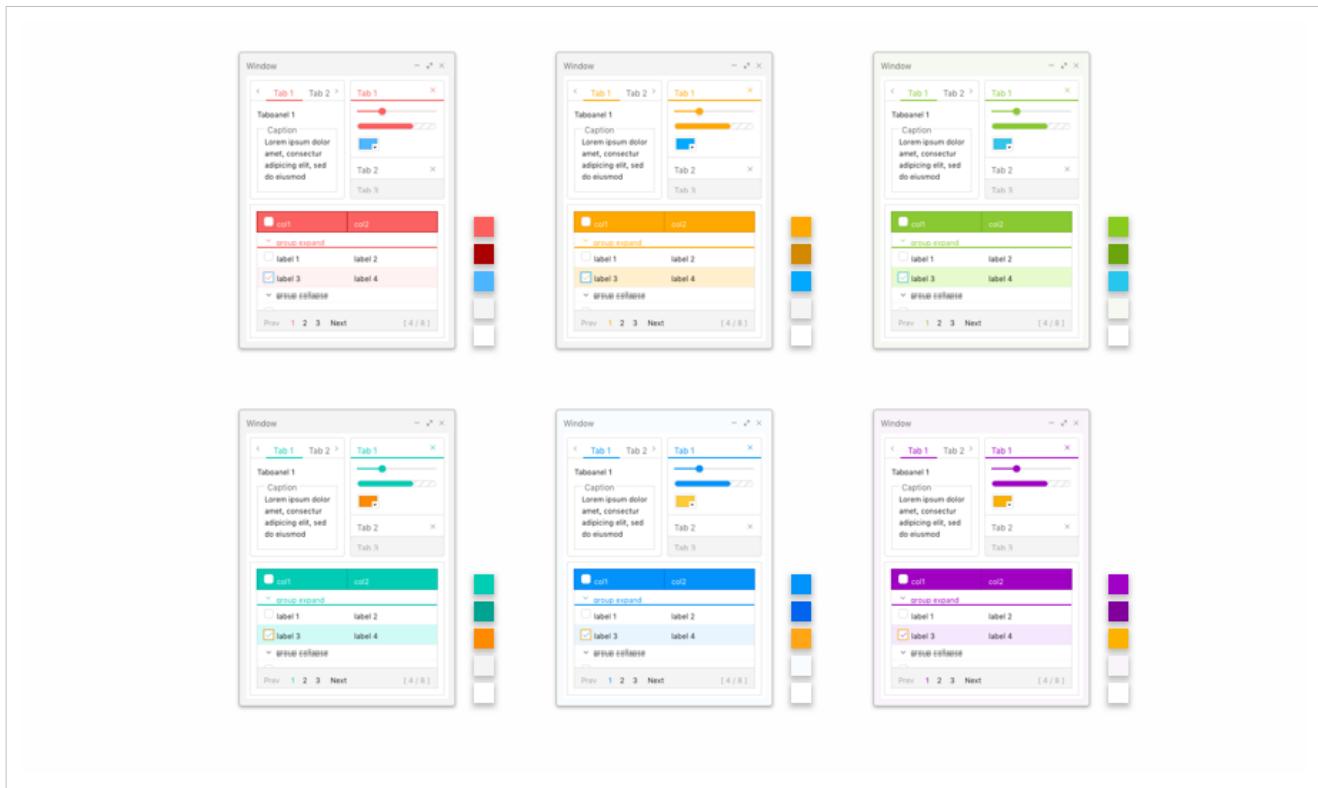
To apply a theme in the theme pack, your project should include the theme pack jar first:

```
<dependency>
  <groupId>org.zkoss.themepack</groupId>
  <artifactId>theme-pack</artifactId>
  <version>${zk.version}</version>
</dependency>
```

Then apply a theme according to ZK Developer's Reference/Theming and Styling/Switching Themes.

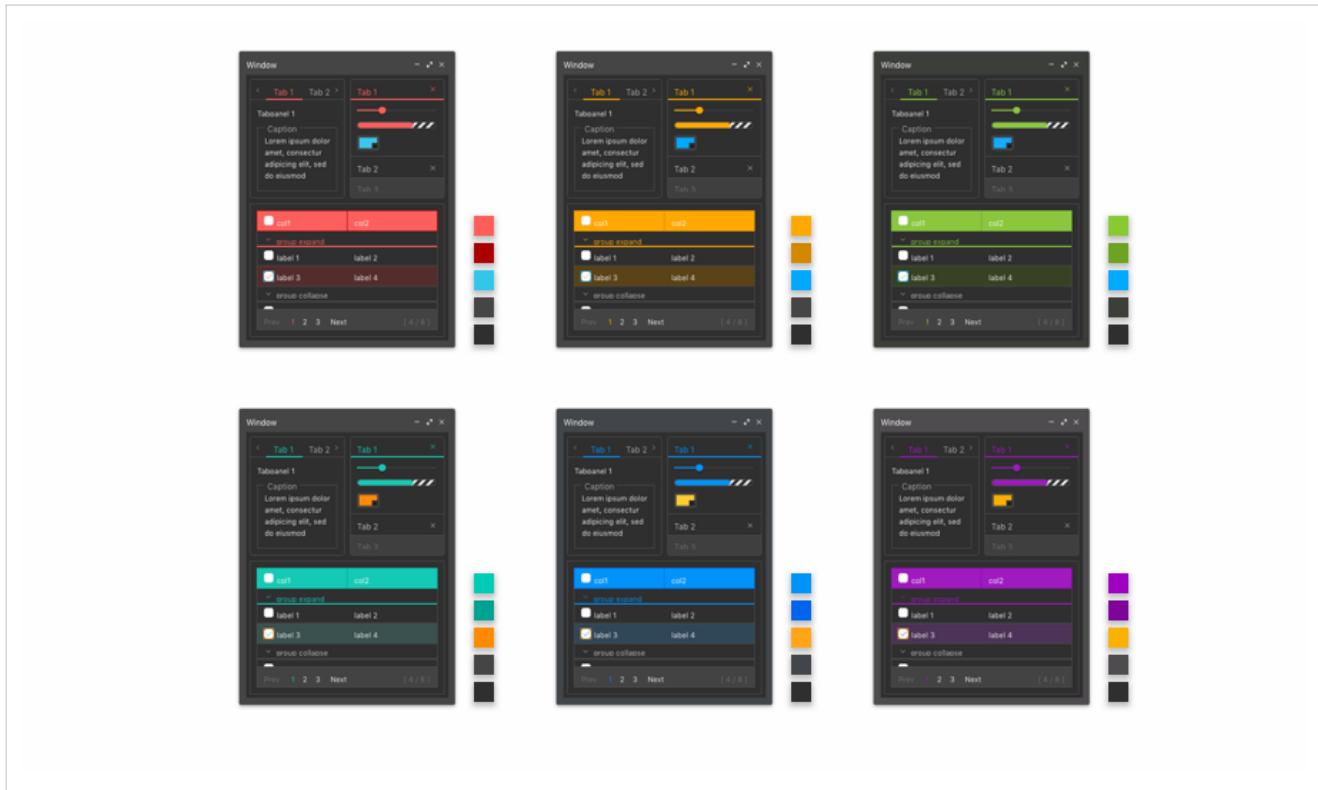
Lite themes

Iceblue(default), Poppy, Marigold, Olive, Aurora, Lavender



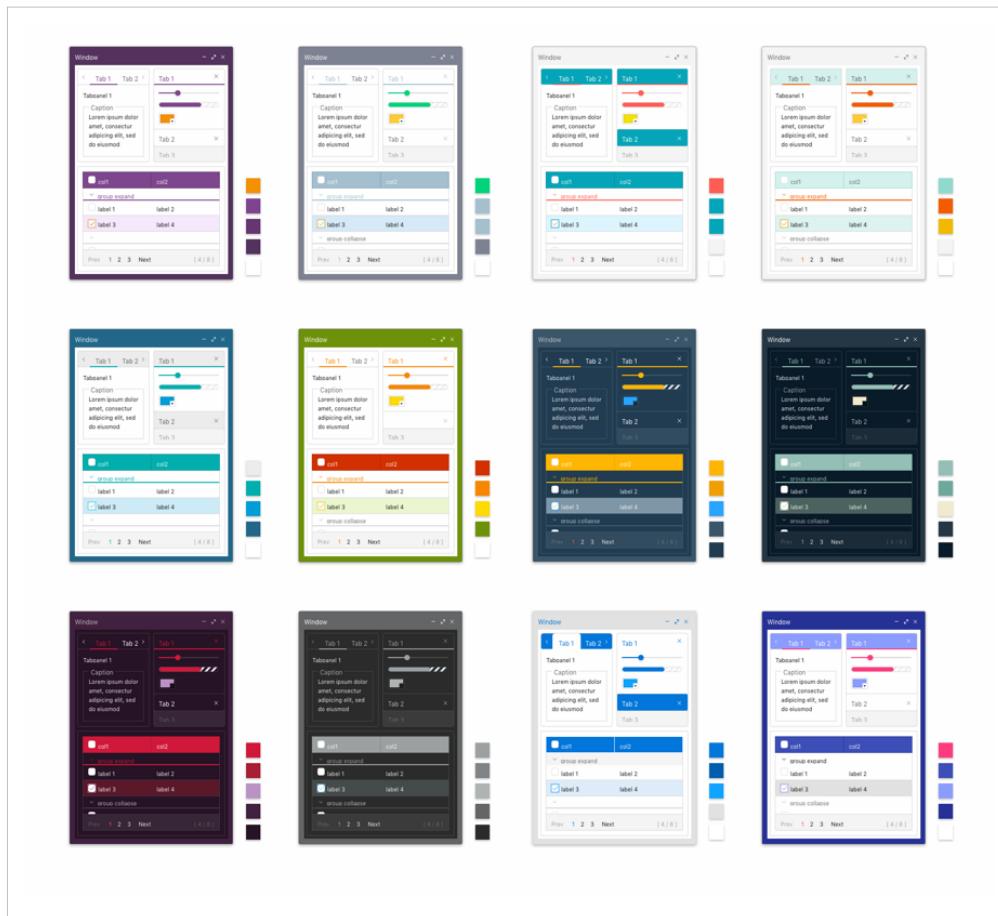
Dark themes

Ruby, Amber, Emerald, Aquamarine, Montana, Violet



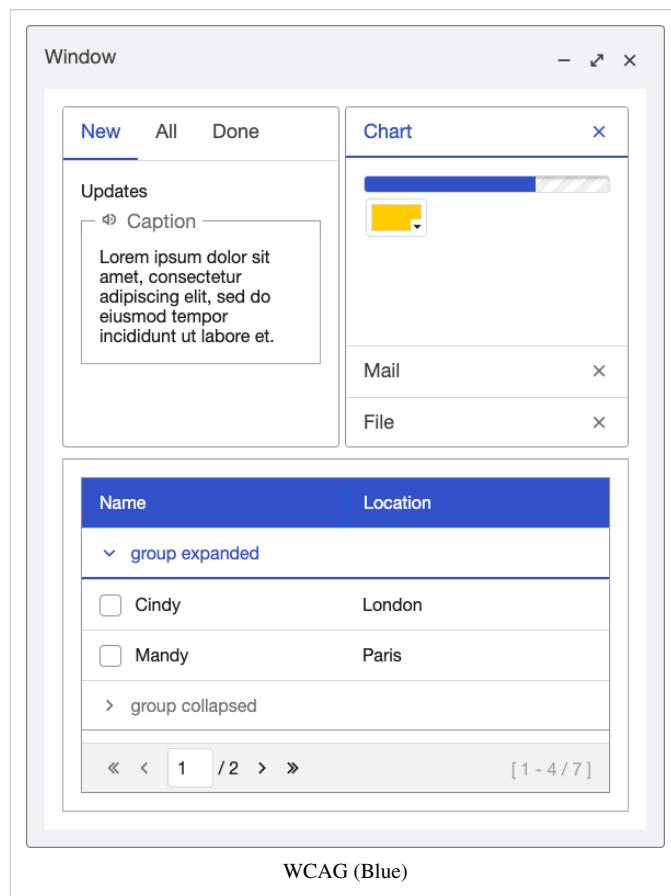
Mix-match

Cheese and Wine, Winter Spring, Blueberry and Raspberry, Macaron, Deep Sea, Garden Salad, Zen, Mysterious Green, Cardinal, Space Black, Office and Material



Accessibility-ready themes

These themes conform to the WCAG2-compliant contrast level and focus styles. Note that for WCAG compliance, in addition to using a WCAG theme, you will also need to include the `za11y` (`zk-accessibility`) module to your project. Read Developer's Reference/Accessibility^[3] for more information.



since 10.0.0) - from github^[5]

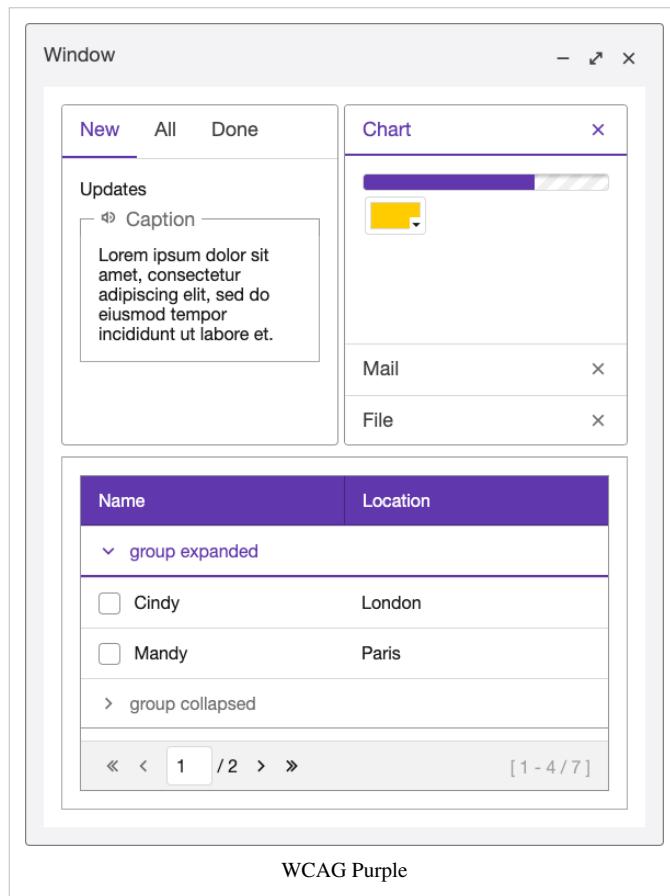
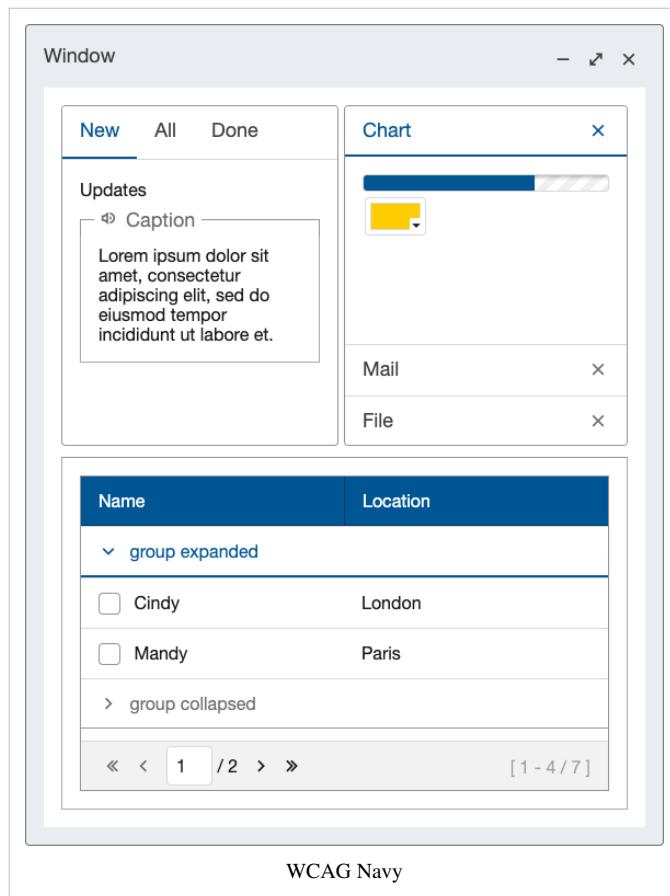
- ZK Theme Pack - download from premium downloads^[6] or ZK EE maven repository^[7].
- 2. Place theme.jar file under "**projectName/WEB-INF/lib**" folder.
- 3. Apply preferred theme by a library property in **zk.xml** file under "**projectName/WEB-INF**" folder

ZK Theme Pack Compact

The breeze-compatible compact variant for ZK Theme Pack Theme is introduced since 8.6.0 allowing developers to upgrade their existing breeze/sapphire/silvertail-themed applications to a modern theme with the minimum code change. Each theme in the theme pack has a corresponding compact theme. The compact themes have smaller font size, padding, margin, but keep the same color design. it's more suitable for migrating from an old theme like breeze without breaking the page layout. Learn more at 8.6 New Features^[4].

Installation

1. Download the preferred theme.jar file or obtain by maven.
 - Default theme (Breeze or Iceblue) - built-in theme, no need to download and register.
 - Sapphire, Silvertail, Atlantic (Deprecated)



```
<!-- zk.xml -->
<library-property>
  <name>org.zkoss.theme.preferred</name>
  <value>sapphire</value> <!-- or silvertail, atlantic, deepsea, gardensalad etc. -->
</library-property>
```

Theme Artifact

```
<dependency>
  <groupId>org.zkoss.theme</groupId>
  <artifactId>breeze</artifactId>
  <version>${zk.version}</version>
</dependency>
<dependency>
  <groupId>org.zkoss.theme</groupId>
  <artifactId>atlantic</artifactId>
  <version>${zk.version}</version>
</dependency>
<dependency>
  <groupId>org.zkoss.theme</groupId>
  <artifactId>sapphire</artifactId>
  <version>${zk.version}</version>
</dependency>
<dependency>
  <groupId>org.zkoss.theme</groupId>
  <artifactId>silvertail</artifactId>
  <version>${zk.version}</version>
</dependency>
<dependency>
  <groupId>org.zkoss.themepack</groupId>      <!-- >=8.6.1 -->
  <!--<groupId>org.zkoss.theme</groupId>--> <!-- <=8.6.0.1 -->
  <artifactId>theme-pack</artifactId>
  <version>${zk.version}</version>
</dependency>
```

All Theme Names of Theme Pack

1. iceblue
2. amber
3. aquamarine
4. aurora
5. blueberryandraspberry
6. cardinal
7. cheeseandwine
8. deepsea
9. emerald
10. gardensalad
11. lavender

12. macaron
13. marigold
14. material
15. montana
16. mysteriousgreen
17. office
18. olive
19. poppy
20. ruby
21. spaceblack
22. violet
23. winterspring
24. zen
25. wcag
26. wcag_navy
27. wcag_purple

Compact Theme Name

Append the name with `_c` gives you the theme name of the compact variant. For example:
 iceblue ---> **iceblue_c**

Switching Themes

You can include multiple themes in the same application and allow your end-users to choose their preferred themes.
 For more information please refer to Switching Themes ^[8]

Customizing a Theme

To build a custom theme based on a standard ZK theme, read Customize a standard theme ^[9].
 Also, reference Theming and Styling ^[10].

Version History

Version	Date	Content
10.0.0	Jan 17, 2023	Deprecate Breeze, Sapphire, Silvertail, and Atlantic

References

- [1] https://www.zkoss.org/wiki/ZK_Developer's_Reference/Theming_and_Styling/ZK_Official_Themes#ZK_Theme_Pack
- [2] <https://www.zkoss.org/zkthemepackdemo/>
- [3] https://www.zkoss.org/wiki/ZK_Developer%27s_Reference/Accessibility
- [4] https://www.zkoss.org/wiki/Small_Talks/2018/November/New_Features_of_ZK_8.6.0#Refresh_Theme_without_Code_Change_-_Compact_Theme
- [5] <http://github.com/zkoss/zkthemes/releases>
- [6] <https://www.zkoss.org/download/premium#zktheme>
- [7] https://www.zkoss.org/wiki/ZK_Installation_Guide/Maven_Setup#PE_.2F_EE_.28premium_users_only.299
- [8] https://www.zkoss.org/wiki/ZK_Developer%27s_Reference/Theming_and_Styling/Switching_Themes
- [9] https://www.zkoss.org/wiki/ZK_Developer%27s_Reference/Theming_and_Styling/Customizing_Standard_Themes

[10] https://www.zkoss.org/wiki/ZK_Developer%27s_Reference/Theming_and_Styling

Switching Themes

Include Theme Jar

Before switching to a non-default theme, please ensure your application **contains the corresponding theme jar** first. Using the default theme doesn't require a separate jar file (it's bundled in ZK framework jars).

For example, since 8.6, the default theme is iceblue. If you want to use **breeze**, you need to include its jar first:

```
<dependency>
    <groupId>org.zkoss.theme</groupId>
    <artifactId>breeze</artifactId>
    <version>${zk.version}</version>
</dependency>
```

Including a theme jar doesn't switch to that theme automatically. You need to explicitly apply the theme mentioned in the following sections.

Theme Resolution Priority

ZK determines(resolves) a theme in the following order:

1. Cookies
2. Library property
3. Theme priority

Apply a Theme for the Whole Application

Specify a theme name as a value. This will apply to the whole application as a default theme.

zk.xml

```
<library-property>
    <name>org.zkoss.theme.preferred</name>
    <value>deepsea</value> <!-- no whitespace in theme names -->
</library-property>
```

Check the complete theme name of the theme pack at [ZK Developer's Reference/Theming and Styling/ZK Official Themes#All Theme Names of Theme Pack](#)

Dynamically switching themes using Cookies

To provide different themes for different end users (sessions), you can use cookies since it's stored in a browser.

```
Themes.setTheme(Executions.getCurrent(), "custom");
Executions.sendRedirect("");
```

Internally, `CookieThemeResolver`^[1] provides this functionality.

Dynamically Switching Themes Using Library Property

Library property is used to apply a preferred theme when the current theme setting could not be obtained from Cookies. Notice that the property change affects the whole application (all end users).

Programmatically:

```
Library.setProperty("org.zkoss.theme.preferred", "custom");
Executions.sendRedirect("");
```

Theme Priority

If the previous two options do not yield any result, the theme with the highest priority would be applied. Theme priority is usually assigned when registering a theme but could also be changed dynamically.

Please refer to `Themes`^[2] for its family of `register()` methods.

Customize the Theme Resolution Process

Web developers could also add other ways for setting the current theme by writing a custom `ThemeResolver`^[3].

If you would like to communicate theme name via session instead, you would create a class like the following:

```
package foo;

public class SessionThemeResolver implements ThemeResolver {
    @Override
    public String getTheme(HttpServletRequest request) {
        Session sess = request.getSession();
        if (sess != null) {
            return sess.getAttribute("mytheme");
        }
    }

    @Override
    public void setTheme(HttpServletRequest request,
        HttpServletResponse response, String themeName) {
        Session sess = request.getSession();
        if (sess != null) {
            sess.setAttribute("mytheme", themeName);
        }
    }
}
```

and configure the custom ThemeResolver in **WEB-INF/zk.xml**.

```
<zk>
    <desktop-config>
        <theme-resolver-class>foo.SessionThemeResolver</theme-resolver-class>
    </desktop-config>
</zk>
```

To access the current theme resolver, please refer to `ThemeFns.getThemeResolver()`^[4] and `ThemeFns.setThemeResolver(org.zkoss.web.theme.ThemeRegistry)`^[5].

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/theme/CookieThemeResolver.html#>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/theme/Themes.html#>
- [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/web/theme/ThemeResolver.html#>
- [4] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/web/fn/ThemeFns.html#getThemeResolver\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/web/fn/ThemeFns.html#getThemeResolver())
- [5] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/web/fn/ThemeFns.html#setThemeResolver\(org.zkoss.web.theme.ThemeRegistry\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/web/fn/ThemeFns.html#setThemeResolver(org.zkoss.web.theme.ThemeRegistry))

Customizing Standard Themes

Here we introduce how to customize the standard themes, such as iceblue, breeze and silver gray.

Customize standard themes using theme template

Please see ZK Developer's Reference/Theeming and Styling/Creating Custom Themes/Theme Template.

Change Font Size and Family

You can use CSS to define the fonts (there are no special config attributes available/required anymore)

If you want to define the default font family **for all ZK components only**:

```
[class*="z-"] :not([class*="z-icon-"]) {
    font-family: Arial;
}
```

If you want to define the default font family for **the whole page body** (including html native element):

```
body * :not([class*="z-icon-"]) {
    font-family: Arial;
}
```

Change Font Size and Family - ZK 6.5 and below

The default theme of ZK components uses the library properties to control the font size and family. You can change them easily by specifying different values.

Notice that the library properties control the theme for the whole application. If you want to provide *per-user* theme (like zkdemo does), you have to implement a theme provider.

Font Size

The default theme uses the following library properties to control the font sizes.

Name	Default	Description
org.zkoss.zul.theme.fontSizeM	12px	The default font size. It is used in the most components.
org.zkoss.zul.theme.fontSizeS	11px	The smaller font size used in the component that requires small fonts, such as toolbar.
org.zkoss.zul.theme.fontSizeXS	10px	The extremely small font size; rarely used.
org.zkoss.zul.theme.fontSizeMS	11px	The font size used in the menu items.

To change the default value, you can specify the library properties in `WEB-INF/zk.xml` as follows.

```
<library-property>
    <name>org.zkoss.zul.theme.fontSizeM</name>
    <value>12px</value>
</library-property>
<library-property>
    <name>org.zkoss.zul.theme.fontSizeS</name>
    <value>10px</value>
</library-property>
<library-property>
    <name>org.zkoss.zul.theme.fontSizeXS</name>
    <value>9px</value>
</library-property>
```

Font Family

The following library properties control the font family.

Name	Description
org.zkoss.zul.theme.fontFamilyT	Default: Verdana, Tahoma, Arial, Helvetica, sans-serif The font family used for titles and captions.
org.zkoss.zul.theme.fontFamilyC	Default: Verdana, Tahoma, Arial, serif The font family used for contents.

Add Additional CSS

If you want to customize certain components, you can provide a CSS file to override the default setting. For example, if you want to customize the look and feel of the `a` component, you can provide a CSS file with the following content.

```
.z-a-disd {  
    color: #C5CACB !important;  
    cursor: default !important;  
    text-decoration: none !important;  
}  
.z-a-disd:visited, .z-a-disd:hover {  
    text-decoration: none !important;  
    cursor: default !important;;  
    border-color: #D0DEF0 !important;  
}
```

Then, specify it in `WEB-INF/zk.xml` as follows.

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

For more information, please refer to the ZK Style Guide.

Creating Custom Themes

ZK provides an extensive component set that allows web developers to use as building blocks for easy web page UI construction. The default styling out-of-the-box gives a look-and-feel that imitates the breeze of the Spring season. While this may suit some perfectly, some may want to have a complete make-over for a unique visual presentation that better matches their sites' thematic flavor or their company's corporate style. In ZK, switching to another 'theme' gives them a way to do just that.

In ZK's term, a **theme** is a collection of stylesheets and associated images for all ZK components.

Please create a new custom theme according to Theme Template.

Themes could be packaged inside a folder. A new theme can be created by first cloning the folder containing an existing theme and then making the necessary changes to the stylesheets and images.

Prior to **6.5.1**, additional themes could only be packaged and made available to web applications inside jar files. Sapphire and silvertail are the two official examples.

Please refer to the subsections according to ZK version you use.

Theme Template

We collect all style-related files into ZK theme template project at Github ^[1], making it much easier to create a custom theme. The main idea here is to have a ZK template theme as the **base theme**, which then allows ZK app developers to fork the repository and commit your changes in your repository to create your new custom theme.

Process

The general steps are:

1. Fork the zkThemeTemplate repository ^[1]
2. initialize the theme project
3. modify LESS ^[2] files and preview
4. build a jar
5. apply to your project

For detailed steps, please see README in zkThemeTemplate ^[1]

Benefits

This approach has some benefits that previous approaches don't have.

Easy to Maintain

Committing your change into a git repository makes your change easy to track in the future. Therefore, your team can know what's difference between your custom theme and ZK standard theme. It's easy to identify an issue.

Easy to Upgrade

If you isolate your custom change into separate files, you can easily merge changes with git from the original repository when there are fixes committed to the original theme template. Moreover, it's can be done automatically. You don't need to manually compare changed files and apply the changes by yourselves. This reduces human errors.

References

[1] <https://github.com/zkoss/zkThemeTemplate>

[2] <https://lesscss.org/>

Archive-based Themes



This documentation is for an older version of ZK. For the latest one, please click here ^[1].

Before creating a new ZK theme, web designers need to understand its directory structure, let's start off by discovering where the default theme (a.k.a. breeze) is. Basically, the default theme is contained inside three java archive files: zul.jar (ZK CE ^[6]), zkex.jar (ZK PE ^[6]) and zkmax.jar (ZK EE ^[6]). **Note:** freshly or evaluation versions will have a special suffix to indicate the zk version and the build date. (e.g. zul-6.5.1.FL.20121204.jar).

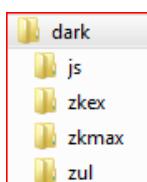
As mentioned previously, a 'theme' is a collection of stylesheets and associated images for ZK's component set. Stylesheets are the files with an extension of ".css.dsp". Think of them as normal CSS files that could utilize JSP taglib functionality. Associated images all have file extension either of ".gif" or ".png".

- Available for ZK:
CE PE EE

In ZK EE, users will also have access to a tablet-enhanced theme in zkmax.jar. In addition to the stylesheets and associated images, the tablet-enhanced theme also contains a property file (**default.theme-properties**) that could be used to easily customize attributes such as font-sizes and color values.

Once those resources are extracted from the respective java archives while preserving the original directory structure, they can become a basis for a new theme, and be ready to be packaged inside a jar file.

The top-level subdirectories for this folder should look similar to the figure below.



Create an Archive-based Theme

Creating an archive-based theme can be broken down into the following steps.

1. Create a theme project skeleton
2. Modify the theme resources

Create a theme project skeleton

The general idea is described in the introductory paragraph. Since sapphire and silvertail^[2] are two official examples of archive-based themes, web developers could simply clone the zkthemes project, and use one of these themes as a starting point. Nonetheless, please note that the two official standard themes are desktop-only themes. If you also want to tailor the view for tablet-clients, you would need the complete set of files that makes up the default theme.

Although manually collecting the files directly from the ZK library files is possible, the process is tedious and error-prone. Hence, this step should be performed using a tool such as the **ZK Default Theme Extractor Utility (ztx.bat)**^[3].

Following are the steps:

1. Download ZK library into a directory.
Note: ZK library can also be found inside an existing ZK project.
2. Execute **ztx.bat** to extract the default theme into an archive

After a typical ztx session has been executed, an archive would be generated that contains the exact replica of the default theme in the folder structure required by ZK theming support. This generated archive becomes the basis where the new theme could be derived.

Since 7.0.0, we provide a maven-archetype that can easily create a ZK theme maven project, refer to the blog^[4].

Modify the theme resources

Now it is just a matter of modifying the relevant stylesheets and importing associated image files.

Instead of creating a theme project from scratch, it is easier to use one of the existing standard theme as a template.

Setting up the environment:

1. Clone zkthemes from github, if haven't done so.
2. Import Sapphire as an Existing Maven Project into Eclipse.
3. Rename all the file names and folder names that contains the word *sapphire* to the theme name of your choice
4. Unpack the generated archive to replace the content originally inside the folder **src/archive/web/sapphire**

Next, the new theme will need to be registered first before it could be used by the ZK application. For archive-based themes, this is done by providing an implementation of the WebAppInit^[5] interface.

Note: the registered name should match the folder name.

For example, assume the custom theme is named **darkstar**,

```
package foo;

public class DarkstarThemeWebAppInit implements WebAppInit {
    @Override
    public void init(WebApp webapp) throws Exception {
        Themes.register("darkstar");
        // Only ZK EE users could use tablet theme
        if ("EE".equals(WebApps.getEdition())) {
```

```
    Themes.register("tablet:darkstar");  
}  
}  
}
```

Also, make sure that **metainfo/zk/config.xml** contained the following configuration.

```
<config>
...
<listener>
    <listener-class>foo.DarkstarThemeWebInit</listner-class>
</listener>
...
</config>
```

Now, the component style modifications shall begin. Please refer to this smalltalk^[6] for a more detailed example on doing this. Even though the smalltalk is about folder-based themes, as far as modifying theme resources is concerned, the procedure is the same.

Here would just summarize the steps.

General steps for component style modification:

1. Locate the stylesheet for a given component
 2. Modify existing images or add new images as needed
 3. Customize the component style by tweaking the stylesheet located in step 1

If a component style rule needs to refer to images within the theme folder, please use the zk core taglib function **encodeThemeURL** for path resolution. For example, to refer to **zul/img/input/combo-btn.png** under the **dark** theme folder, use the following syntax.

```
<%@ taglib uri="http://www.zkoss.org/dsp/web/core" prefix="c" %>  
...  
.z-combobox {  
    background-image:  
url(${c:encodeThemeURL('~/zul/img/input/combo-btn.png')});  
}
```

Note: The special prefix `~./` will be interpreted as the theme folder root (e.g. `/theme/dark/`).

After all this has been done, the components should have their views customized. Please refer to this article [\[7\]](#) for how to switch themes dynamically within the ZK application.

- Available for ZK:

Developers could also follow the same process described above to tailor the appearance of ZK components when viewed on tablets. When locating the stylesheets to modify, look inside the `~/zkmax/css/tablet` folder instead.

For ZK EE users, custom themes could support styling for desktop-only, tablet-only, or both. Web application needs to know about the platforms a custom theme may support. This is also accomplished through theme registration. When a custom theme overrides the default tablet theme, its theme name must be prefixed with "**tablet:**" *before making registration. For example, to notify the web application that 'dark' theme is tablet-capable, please use the following code snippet.*

```
Themes.register("tablet:dark");
```

In addition, the default tablet-enhanced theme has refactored many attributes such as color values, font-sizes, border-widths, ... into a property file in `~/zkmax/default.theme-properties`. Changing the attribute values inside this property file could quickly alter the appearance of the components without touching their stylesheets. Nevertheless, developers may also combine these two approaches to suit their needs.

After the new theme is developed, the theme project can be packaged as a jar file for distribution, say **darkstar.jar**. This could be done by executing the command **mvn clean package** at the project root.

Use an Archive-based Theme

Using an archive-based theme in a ZK Application is simple. Simply put the theme jar file inside the **WEB-INF/lib** folder of your ZK application. During the startup of your application, the new custom theme would be automatically registered, and available to use.

The process can be summarized as follows:

1. Put the theme jar file inside **WEB-INF/lib** folder
2. Ready to use

References

- [1] https://www.zkoss.org/wiki/ZK_Developer%27s_Reference/Theming_and_Styling/Creating_Custom_Themes/Theme_Template
- [2] ZK Themes (<https://github.com/zkoss/zkthemes>)
- [3] ZK Default Theme Extractor Utility. Please download at github (<https://gist.github.com/raw/4334775/e5d669bb873443aa03f8febffcc3fc4b2518ecb/ztx.bat>).
- [4] <http://blog.zkoss.org/index.php/2013/09/17/zk7-create-a-new-a-theme-project/>
- [5] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/WebAppInit.html#>
- [6] http://http://books.zkoss.org/index.php?title=Small_Talks/2013/January/Packaging_Themes_Inside_Folders_in_ZK_6.5.2
- [7] http://books.zkoss.org/wiki/ZK_Developer%27s_Reference/Theming_and_Styling/Understanding_the_Theming_Subsystem/Switching_Themes

Folder-based Themes



This documentation is for an older version of ZK. For the latest one, please click here ^[1].

Before creating a new ZK theme, web designers need to understand its directory structure, let's start off by discovering where the default theme (a.k.a. breeze) is. Basically, the default theme is contained inside three java archive files: zul.jar (ZK CE ^[6]), zkex.jar (ZK PE ^[6]) and zkmax.jar (ZK EE ^[6]). **Note:** freshly or evaluation versions will have special suffix to indicate the zk version and the build date. (e.g. zul-6.5.1.FL.20121204.jar).

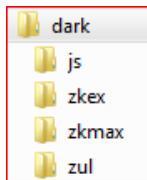
As mentioned previously, a 'theme' is a collection of stylesheets and associated images for ZK's component set. Stylesheets are the files with extension of ".css.dsp". Think of them as normal CSS files that could utilize JSP taglib functionality. Associated images all have file extension either of ".gif" or ".png".

- Available for ZK:
 - CE
 - PE
 - **EE**

In ZK EE, users will also have access to tablet-enhanced theme in zkmax.jar. In addition to the stylesheets and associated images, the tablet-enhanced theme also contains a property file (**default.theme-properties**) that could be used to easily customize attributes such as font-sizes and color values.

Once those resources are extracted from the respective java archives while preserving the original directory structure, they can be placed inside a folder, and become a basis for a new theme.

The top level subdirectories for this folder should look similar to the figure below.



Create a Folder-based Theme

Introduced in ZK 6.5.2, the embodiment of a theme can come from a sub-folder under the web application's context root. Creating a folder-based theme can be broken down into the following steps.

1. Create a theme folder skeleton
2. Modify the theme resources

Create a theme folder skeleton

The general idea is described in the introductory paragraph. However, it is tedious and error-prone to do this step manually. Hence, these steps should be performed using a tool such as the **ZK Default Theme Extractor Utility (ztx.bat)**^[1].

Following are the steps:

1. Download ZK library into a directory.
Note: ZK library can also be found inside an existing ZK project.
2. Execute **ztx.bat** to extract the default theme into an archive

After a typical ztx session has been executed, an archive would be generated that contains the exact replica of the default theme in the folder structure required by ZK theming support. This generated archive becomes the basis where the new theme could be derived.

Modify the theme resources

Now it is just a matter of modifying the relevant stylesheets and importing associated image files.

Setting up the environment:

1. Create a ZK Application Project [2]
2. Create a folder named **theme** under the root folder of the web content.
3. Unpack the generated archive under the folder **theme**

Also, please make sure the ZK Application is configured to process *.css.dsp by the following configuration in **WEB-INF/web.xml**.

```
<servlet>
    <servlet-name>dspLoader</servlet-name>
    <servlet-class>org.zkoss.web.servlet.dsp.InterpreterServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>dspLoader</servlet-name>
    <url-pattern>*.dsp</url-pattern>
</servlet-mapping>
```

Next, the new theme will need to be registered first before it could be used by the ZK application. Since the origin of the new theme is from a folder, ZK 6.5.2 extends the theme registration API for this purpose. **ThemeOrigin** is an enum defined to specify the origin of the registered theme. It has two valid values: **JAR** (default) and **FOLDER**. Since **ThemeOrigin.JAR** is the default value, the extended theme registration API is only needed in the case of folder-based themes. Theme registration could be done in the initialization code of the view model.

Note: the registered name should match the folder name.

For example,

```
...
import org.zkoss.zul.theme.Themes;
import org.zkoss.web.theme.StandardTheme.ThemeOrigin;
...
public class MainViewModel {
    ...
    @Init
    public void init() {
        ...
        Themes.register("dark", ThemeOrigin.FOLDER);
        ...
    }
    ...
}
```

Now, the component style modifications shall begin. Please refer to this smalltalk for a more detailed example on doing this. Here would just summarize the steps.

General steps for component style modification:

1. Locate the stylesheet for a given component
2. Modify existing images or add new images as needed
3. Customize the component style by tweaking the stylesheet located in step 1

If a component style rule needs to refer to images within the theme folder, please use the zk core taglib function **encodeThemeURL** for path resolution. For example, to refer to **zul/img/input/combo-btn.png** under the **dark** theme folder, use the following syntax.

```
<%@ taglib uri="http://www.zkoss.org/dsp/web/core" prefix="c" %>
...
.z-combobox {
    background-image:
url(${c:encodeThemeURL('~/zul/img/input/combo-btn.png')});
}
```

Note: The special prefix `~./` will be interpreted as the theme folder root (e.g. `/theme/dark/`).

After all this has been done, the components should have their views customized. Please refer to this article ^[7] for how to switch themes dynamically within the ZK application.

- Available for ZK:
- **CE** **PE** **EE**

Developers could also follow the same process described above to tailor the appearance of ZK components when viewed on tablets. When locating the stylesheets to modify, look inside the `~/zkmax/css/tablet` folder instead.

For ZK EE users, custom themes could support styling for desktop-only, tablet-only, or both. Web application needs to know about the platforms a custom theme may support. This is also accomplished through theme registration. When a custom theme overrides the default tablet theme, its theme name must be prefixed with `"tablet:"` before making registration. For example, to notify the web application that **dark** theme is tablet-capable, please use the following code snippet.

```
Themes.register("tablet:dark", ThemeOrigin.FOLDER);
```

In addition, the default tablet-enhanced theme has refactored many attributes such as color values, font-sizes, border-widths, ... into a property file in `~/zkmax/default.theme-properties`. Changing the attribute values inside this property file could quickly alter the appearance of the components without touching their stylesheets. Nevertheless, developers may also combine these two approaches to suit their needs.

After the new theme is developed, the entire theme folder can be exported as a zip file for distribution, say **dark.zip**.

Use a Folder-based Theme

Using a folder-based theme in a ZK Application is simple and versatile. Simply adopt the same environment as the one for developing a new folder-based theme. Furthermore, the ZK Application must be informed of the existence of the newly installed theme.

The process can be summarized as follows:

1. Create a theme root folder
2. Install the folder-based theme
3. Register the folder-based theme

Let's walk through an example of using the folder-based theme **dark.zip** in another ZK application

Create a theme root folder

Theme root folder is where a ZK Application stores all its folder-based themes. By default, this folder is assumed to be named 'theme' and is directly under the application's root directory for its web content. This default location can be changed via the library property **org.zkoss.theme.folder.root**. For example, to move the theme root folder to **/view/themes**, the web developer would make the following configuration setting in **WEB-INF/zk.xml**. Please note that the value for the theme root folder cannot have leading and trailing forward slashes.

```
<library-property>
    <name>org.zkoss.theme.folder.root</name>
    <value>view/themes</value>
</library-property>
```

Install the folder-based theme

Suppose the theme root folder is changed to **/view/themes** through the configuration setting and that directory has been created. Simply extract the theme folder under this directory would finish this step. Since the name of the theme folder is also the name of the theme, renaming the theme folder would also rename the theme. For instance, to change the theme name from **dark** to **darkstar**, one would rename the folder accordingly.

Register the folder-based theme

Before the folder-based theme can be used, it must be registered first. The relevant code is as follows.

```
Themes.register("darkstar", ThemeOrigin.FOLDER);
// For ZK EE, also make customized tablet theme available
if ("EE".equals(WebApps.getEdition()))
    Themes.register("tablet:darkstar", ThemeOrigin.FOLDER);
```

This code fragment can be written in several places. For an example incorporating MVVM, please refer to the section [Modify the theme resource](#). To make the folder-based theme available at application startup, write a class implementing the [WebAppInit](#)^[3] interface and place the above code inside the init() function.

```
public class DarkstarThemeWebAppInit implements WebAppInit {
    public void init(WebApp webapp) throws Exception {
        Themes.register("darkstar", ThemeOrigin.FOLDER);
        // For ZK EE, also make customized tablet theme available
        if ("EE".equals(WebApps.getEdition()))
            Themes.register("tablet:darkstar", ThemeOrigin.FOLDER);
    }
}
```

The configuration file **WEB-INF/zk.xml** must also include the following configuration item.

```
<listener>
    <listener-class>DarkstarThemeWebAppInit</listener-class>
</listener>
<library-property>
    <name>org.zkoss.theme.preferred</name>
    <value>darkstar</value>
</library-property>
```

References

- [1] ZK Default Theme Extractor Utility. Please download at github (<https://gist.github.com/raw/4334775/e5d669bb873443aa03f8febffcc3fc4b2518ecb/ztx.bat>).
- [2] Please refer to ZK Installation Guide (http://books.zkoss.org/wiki/ZK_Installation_Guide)
- [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/WebAppInit.html>

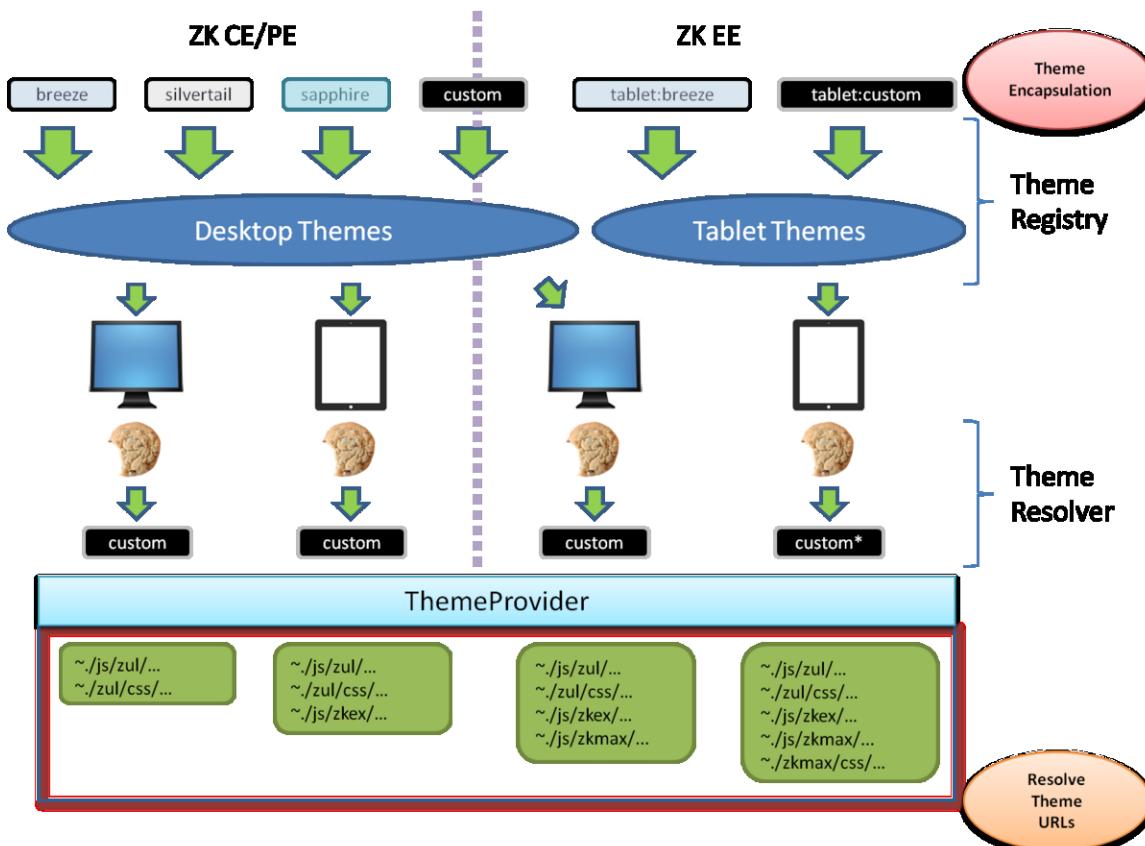
Understanding the Theming Subsystem

The collection of the components' stylesheets and associated images is called a theme. Themes control the overall look and feel of a page composed of ZK components. For example, the components using the standard sapphire theme will all follow the same blue-ish color scheme.

Starting from ZK 6.5.2, supporting different themes takes a more modularized approach. **Theme Support Subsystem** breaks up into a few pluggable modules: **ThemeRegistry**, **ThemeResolver**, and **ThemeProvider**. ThemeRegistry keeps track of a list of available themes to use. ThemeResolver is responsible for setting and obtaining the current theme. ThemeProvider allows web developers to manipulate the CSS stylesheets actually employed to style the UI components. Customizing these modules enables web developers to modify any part of the theme processing pipeline to fit their particular requirement.

Theme Support Subsystem is illustrated by the figure below. In addition to the three pluggable modules, there also exist facilities to encapsulate theme information and resolve URLs for locating theme resources.

Please refer to the subsections for more information on the constituent parts.



Information about a Theme

Apart from having a name, a theme could be associated with many attributes. Encapsulating theme-specific attributes is defined in `Theme`^[1]. Each theme should have at least a name, which helps the web application to identify it. Web developers should extend this abstract class to define other attributes associated with concrete themes, such as file paths included in a theme, or variables that could be used to parameterize a theme.

Standard themes have additional attributes like a more descriptive name for displaying purposes, a priority value to help the system choose the theme to use, and an origin of the theme's resources (i.e. CSS and image files). Standard theme information is provided by `StandardTheme`^[2].

Example

```
import org.zkoss.web.theme.Theme;

public class MyTheme extends Theme {
    // theme-specific attributes
    ...
    // getters/setters
    ...
}
```

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/web/theme/Theme.html#>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/web/theme/StandardTheme.html#>

Registering your Theme

Before using a theme, it must be registered so that the system knows about its existence and where to retrieve its resources (from a jar file or from a folder). Themes^[2] provides several static methods for registering your themes.

- Available for ZK:

• **CE PE EE**

Registration, in its simplest form, is to tell the web application about the name of the theme. It will be assumed that the theme is for desktop only, and its resources should be retrieved from a jar file. For example,

```
Themes.register("custom");
```

- Available for ZK:

• **CE PE EE**

Starting from ZK 6.5.2, theme resources could also be retrieved from a folder. To indicate that a theme is folder-based, please use ThemeOrigin^[1] to specify the origin of the theme resources, like below.

```
Themes.register("custom", ThemeOrigin.FOLDER);
```

- Available for ZK:

• **CE PE EE**

In ZK EE 6.5.0+, components would appear differently when viewed on tablet devices. A custom theme could be applicable to desktop only, tablet only, or both. To signify that a theme also serves tablet devices, please attach a "tablet:" prefix in front of the theme name when registering. For example,

```
Themes.register("tablet:custom");
```

Creating a Custom Theme Registration Service

- Available for ZK:

• **CE PE EE**

ThemeRegistry^[2] defines the interface to create a repository of themes available to the web application. DesktopThemeRegistry^[3] (for ZK CE/PE) and ResponsiveThemeRegistry^[4] (for ZK EE) are the standard implementations that actually store the registered themes.

Standard theme registries would accept all theme registrations. Duplicate registration would update theme information. Registered themes are available to all users. (For ZK EE, desktop clients would only have desktop themes available, and tablet clients would only have tablet themes available.) If you would like to modify any of these behaviors, please provide a custom ThemeRegistry.

For example, a custom ThemeRegistry could be created by implementing ThemeRegistry^[2] interface directly, or extending one of the standard implementations, depending on the ZK edition you are using.

```
package foo;

public class CustomThemeRegistry implements ThemeRegistry {
    ...
}
```

And then, configure the custom ThemeRegistry in **WEB-INF/zk.xml**.

```
<zk>
    <desktop-config>
        <theme-registry-class>foo.CustomThemeRegistry</theme-registry-class>
    </desktop-config>
</zk>
```

To access the current theme registry, please refer to `ThemeFns.getThemeRegistry()`^[5] and `ThemeFns.setThemeRegistry(org.zkoss.web.theme.ThemeRegistry)`^[6].

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/web/theme/StandardTheme/ThemeOrigin.html#>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/web/theme/ThemeRegistry.html#>
- [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/theme/DesktopThemeRegistry.html#>
- [4] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkmax/theme/ResponsiveThemeRegistry.html#>
- [5] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/web/fn/ThemeFns.html#setThemeRegistry\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/web/fn/ThemeFns.html#setThemeRegistry())
- [6] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/web/fn/ThemeFns.html#setThemeRegistry\(org.zkoss.web.theme.ThemeRegistry\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/web/fn/ThemeFns.html#setThemeRegistry(org.zkoss.web.theme.ThemeRegistry))

Providing Theme Resources

ThemeProvider

After switching to another theme, a `ThemeProvider`^[1] allows you the full control of CSS styling, including but not limited to

- Switch among multiple sets of stylesheets based on, say, the user's preference, cookie, locale, or others
- Replace the CSS styling of component(s) with your own custom styling
- Inject additional CSS files

Standard implementations of `ThemeProvider` are available for each of ZK editions (CE, PE, and EE).

Edition	ThemeProvider
ZK CE	<code>org.zkoss.zul.theme.StandardThemeProvider</code> ^[2]
ZK PE	<code>org.zkoss.zkex.theme.StandardThemeProvider</code> ^[3]
ZK EE	<code>org.zkoss.zkmax.theme.StandardThemeProvider</code> ^[4]

We will illustrate the theme provider with two examples. One is straightforward: set the corresponding attributes based on the cookie. The other **injects** a fragment to the URI such that we can allow the browser to cache the CSS file.

For information of 3.6 and earlier, please refer to ZK 3 Theme Provider.

Examples

A Simple Example

In the following example, we store the preferred font size and the skin (theme) in the cookie and retrieve them when required.

Since 7.0.0, the font size attributes are deprecated because of using LESS.

```
package my;
public class MyThemeProvider implements ThemeProvder {
    public Collection getThemeURIs(Execution exec, List uris) {
        if ("silvergray".equals(getSkinCookie(exec))) {
            uris.add("./silvergray/color.css.dsp");
            uris.add("./silvergray/img.css.dsp");
        }
        return uris;
    }
    public int getWCSCacheControl(Execution exec, String uri) {
        return -1;
    }
    public String beforeWCS(Execution exec, String uri) {
        final String fsc = getFontSizeCookie(exec);
        if ("lg".equals(fsc)) {
            exec.setAttribute("fontSizeM", "15px");
            exec.setAttribute("fontSizeMS", "13px");
            exec.setAttribute("fontSizes", "13px");
            exec.setAttribute("fontSizeXS", "12px");
        } else if ("sm".equals(fsc)) {
            exec.setAttribute("fontSizeM", "10px");
            exec.setAttribute("fontSizeMS", "9px");
            exec.setAttribute("fontSizes", "9px");
            exec.setAttribute("fontSizeXS", "8px");
        }
        return uri;
    }
    public String beforeWidgetCSS(Execution exec, String uri) {
        return uri;
    }
    /** Returns the font size specified in cookie. */
    private static String getFontSizeCookie(Execution exec) {
        Cookie[] cookies =
((HttpServletRequest)exec.getNativeRequest()).getCookies();
        if (cookies!=null)
            for (int i=0; i<cookies.length; i++)
                if ("myfontsize".equals(cookies[i].getName()))

```

```

        return cookies[i].getValue();
    return "";
}
/** Returns the skin specified in cookie. */
private static String getSkinCookie(Execution exec) {
    Cookie[] cookies =
((HttpServletRequest)exec.getNativeRequest()).getCookies();
    if (cookies!=null)
        for (int i=0; i<cookies.length; i++)
            if ("myskin".equals(cookies[i].getName()))
                return cookies[i].getValue();
    return "";
}
}

```

Notice that we return -1 when `java.lang.String`) `ThemeProvider.getWCSCacheControl(org.zkoss.zk.ui.Execution, java.lang.String)`^[5] is called to disallow the browser to cache the CSS file. It is necessary since we manipulate the content of the CSS file by setting the attributes (based on the cookie). It means the content might be different with the same request URL. For a cacheable example, please refer to the next section.

Then, you configure `WEB-INF/zk.xml` by adding the following lines.

```

<desktop-config>
    <theme-provider-class>my.MyThemeProvider</theme-provider-class>
</desktop-config>

```

A Cacheable Example

To improve the performance, it is better to allow the browser to cache the CSS file as often as possible. With the theme provider, it can be done by returning a positive number when `java.lang.String`) `ThemeProvider.getWCSCacheControl(org.zkoss.zk.ui.Execution, java.lang.String)`^[5] is called. However, because the browser will use the cached version, we have to ensure the browser gets a different URL for each different theme. Here we illustrate a technique called **fragment injection**.

The idea is simple: when `java.util.List`) `ThemeProvider.getThemeURIs(org.zkoss.zk.ui.Execution, java.util.List)`^[6] is called, we **inject** a special fragment to denote the content, such that each different theme is represented with a different URL. The injection can be done easily with the inner class called `Aide`^[7]. For example,

```

final String fsc = getFontSizeCookie(exec);
if (fsc != null && fsc.length() > 0) {
    for (ListIterator it = uris.listIterator(); it.hasNext();) {
        final String uri = (String)it.next();
        if (uri.startsWith(DEFAULT_WCS)) {
            it.set(Aide.injectURI(uri, fsc));
            break;
        }
    }
}

```

Then, we can retrieve the fragment we encoded into the URI later when `java.lang.String`) `ThemeProvider.beforeWCS(org.zkoss.zk.ui.Execution, java.lang.String)`^[8] is called. It can be done easily by use of

`Aide.decodeURI(java.lang.String)` [9]. `Aide.decodeURI(java.lang.String)` [9] returns a two-element array if the fragment is found. The first element is the URI without fragment, and the second element is the fragment. For example,

```
public String beforeWCS(Execution exec, String uri) {
    final String[] dec = Aide.decodeURI(uri);
    if (dec != null) {
        if ("lg".equals(dec[1])) {
            exec.setAttribute("fontSizeM", "15px");
            exec.setAttribute("fontSizeMS", "13px");
            exec.setAttribute("fontSizes", "13px");
            exec.setAttribute("fontSizeXS", "12px");
        } else if ("sm".equals(dec[1])) {
            exec.setAttribute("fontSizeM", "10px");
            exec.setAttribute("fontSizeMS", "9px");
            exec.setAttribute("fontSizes", "9px");
            exec.setAttribute("fontSizeXS", "8px");
        }
        return dec[0];
    }
    return uri;
}
```

Here is a complete example:

```
public class CacheableThemeProvider implements ThemeProvider{
    private static String DEFAULT_WCS = "~./zul/css/zk.wcs";

    public Collection getThemeURIs(Execution exec, List uris) {
        //font-size
        final String fsc = getFontSizeCookie(exec);
        if (fsc != null && fsc.length() > 0) {
            for (ListIterator it = uris.listIterator();
it.hasNext();) {
                final String uri = (String)it.next();
                if (uri.startsWith(DEFAULT_WCS)) {
                    it.set(Aide.injectURI(uri, fsc));
                    break;
                }
            }
        }

        //silvergray
        if ("silvergray".equals(getSkinCookie(exec))) {
            uris.add("~./silvergray/color.css.dsp");
            uris.add("~./silvergray/img.css.dsp");
        }
        return uris;
    }
}
```

```
public int getWCSCacheControl(Execution exec, String uri) {
    return 8760; //safe to cache
}
public String beforeWCS(Execution exec, String uri) {
    final String[] dec = Aide.decodeURI(uri);
    if (dec != null) {
        if ("lg".equals(dec[1])) {
            exec.setAttribute("fontSizeM", "15px");
            exec.setAttribute("fontSizeMS", "13px");
            exec.setAttribute("fontSizeS", "13px");
            exec.setAttribute("fontSizeXS", "12px");
        } else if ("sm".equals(dec[1])) {
            exec.setAttribute("fontSizeM", "10px");
            exec.setAttribute("fontSizeMS", "9px");
            exec.setAttribute("fontSizeS", "9px");
            exec.setAttribute("fontSizeXS", "8px");
        }
        return dec[0];
    }
    return uri;
}

public String beforeWidgetCSS(Execution exec, String uri) {
    return uri;
}

/** Returns the font size specified in cookie. */
private static String getFontSizeCookie(Execution exec) {
    Cookie[] cookies =
((HttpServletRequest)exec.getNativeRequest()).getCookies();
    if (cookies!=null)
        for (int i=0; i<cookies.length; i++)
            if ("myfontsize".equals(cookies[i].getName()))
                return cookies[i].getValue();
    return "";
}
/** Returns the skin specified in cookie. */
private static String getSkinCookie(Execution exec) {
    Cookie[] cookies =
((HttpServletRequest)exec.getNativeRequest()).getCookies();
    if (cookies!=null)
        for (int i=0; i<cookies.length; i++)
            if ("myskin".equals(cookies[i].getName()))
                return cookies[i].getValue();
    return "";
}
```

```
}
```

How to Specify the Media Types

In addition to String instances, you can return instances of StyleSheet [10] in the returned collection of ThemeProvider.getThemeURIs(org.zkoss.zk.ui.Execution.java.util.List) [11], such that you can control more about the generated CSS link. For example, if you want to add a CSS link for the media type [12], say, print, handheld, then you can do as follows.

```
public Collection getThemeURIs(Execution exec, List uris) {
    uris.add(new StyleSheet("/theme/foo.css", "text/css", "print,
handheld", false));
    return uris;
}
```

Version History

Version	Date	Content
5.0.3	June 2010	The media type was allowed in StyleSheet [10].

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ThemeProvider.html#>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/theme/StandardThemeProvider.html>
- [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkex/theme/StandardThemeProvider.html>
- [4] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkmax/theme/StandardThemeProvider.html>
- [5] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ThemeProvider.html#getWCSCacheControl\(org.zkoss.zk.ui.Execution,java.util.List\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ThemeProvider.html#getWCSCacheControl(org.zkoss.zk.ui.Execution,java.util.List))
- [6] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ThemeProvider.html#getThemeURIs\(org.zkoss.zk.ui.Execution,java.util.List\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ThemeProvider.html#getThemeURIs(org.zkoss.zk.ui.Execution,java.util.List))
- [7] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ThemeProvider/Aide.html#>
- [8] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ThemeProvider.html#beforeWCS\(org.zkoss.zk.ui.Execution,java.util.List\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ThemeProvider.html#beforeWCS(org.zkoss.zk.ui.Execution,java.util.List))
- [9] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ThemeProvider/Aide.html#decodeURI\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ThemeProvider/Aide.html#decodeURI(java.lang.String))
- [10] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/web/servlet/StyleSheet.html#>
- [11] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ThemeProvider.html#getThemeURIs\(org.zkoss.zk.ui.Execution,java.util.List\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ThemeProvider.html#getThemeURIs(org.zkoss.zk.ui.Execution,java.util.List))
- [12] <http://www.w3.org/TR/CSS2/media.html>

Resolving Theme URLs

Themes comprises of stylesheets and images. The URLs for those resources must be resolved once the theme changes.

Several APIs were available to redirect theme resources to the correct location.

ServletFns.encodeThemeURL(java.lang.String)^[1] is for translating image locations inside *.css.dsp files.

ServletFns.resolveThemeURL(java.lang.String)^[2] is for redirecting the retrieval of stylesheets inside a ThemeProvider.

Example Usage (inside *.css.dsp)

```
tr.z-row-over > td.z-row-inner, tr.z-row-over > .z-cell {  
    background-image:  
url(${c:encodeThemeURL('~/zul/img/grid/column-over.png')});  
}
```

Example Usage (inside a ThemeProvider):

```
...  
public String beforeWidgetCSS(Execution exec, String uri) {  
    if (uri.startsWith("~/zul/css/") ||  
        uri.startsWith("~/js/zul/")) {  
  
        uri = ServletFns.resolveThemeURL(uri);  
    }  
  
    return uri;  
}  
...
```

References

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

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

Internationalization

This chapter describes how to make ZK applications flexible enough to run in any locale.

First of all, ZK enables developers to embed Java code and EL expressions any way you like. You could use any Internationalization method you want, such as `java.util.ResourceBundle`.

However, ZK has some built-in support of internationalization that you might find them useful.

Locale

Overview

The locale used to process requests and events is, by default, determined by the browser's preferences (by `javax.servlet.ServletRequest.getLocale()`). For example, `DE` is assumed if a user is using a DE-version browser (unless they changed the setting).

In this section, we'd like to introduce how to configure ZK to handle the locale differently. For example, you can configure ZK to use the same Locale for all users no matter how the browser is configured. Another example is that you can configure ZK to use the preferred locale that a user specifies in his or her profile if you maintain the user profiles in the server.

Current Locale

`Locales.getCurrent()`^[1] returns the current locale that ZK detected in the below precedence.

The Decision Sequence of Locale

ZK determines the current locale in the following sequence:

1. It checks if an attribute called `org.zkoss.web.preferred.locale` defined in the HTTP session (or Session^[2]). If so, use it.
2. It checks if an attribute called `org.zkoss.web.preferred.locale` defined in the Servlet context (or Application^[2]). If so, use it.
3. It checks if a property called `org.zkoss.web.preferred.locale` defined in the library property (i.e., Library^[3]). If so, use it.
4. If none of them is found, it uses the locale defined in the Servlet request (i.e., `ServletRequest.getLocale()`). This is determined by the browser language setting^[4].

With this sequence in mind, you could configure ZK to use the correct locale based on the application requirements.

Application-level Locale

If you want to use the same locale for all users, you can specify the locale in the library property. For example, you could specify the following in `WEB-INF/zk.xml`:

```
<library-property>
  <name>org.zkoss.web.preferred.locale</name>
  <value>de</value>
</library-property>
```

Alternatively, if you prefer to specify it in Java, you could invoke `java.lang.String`) `Library.setProperty(java.lang.String, java.lang.String)`^[5]. Furthermore, to avoid typos, you could use `Attributes.PREFERRED_LOCALE`^[6] as follows.

```
Library.setProperty(Attributes.PREFERRED_LOCALE, "de");
```

Per-user Locale

Because ZK will check if there is a session attribute for the default locale, you could configure ZK to have a per-user locale by specifying the attribute in a session.

For example, you can do this when a user logs in.

```
import org.zkoss.web.Attributes;
...
void login(String username, String password) {
    //check password
    ...
    Locale preferredLocale = ...; //decide the locale (from, say,
database)
    session.setAttribute(Attributes.PREFERRED_LOCALE,
preferredLocale);
    ...
}
```

The Request Interceptor

Deciding the locale after the user logs in may be a bit late for some applications. For example, you might want to use the same Locale that was used in the previous session, before the user logs in. For a Web application, it is usually done by storing the information in a cookie. It can be done by registering a request interceptor, and then manipulating the cookies when the interceptor is called.

A request interceptor is used to intercept each request being processed. It must implement the `RequestInterceptor`^[7] interface. For example,

```
import java.util.Locale;

import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;

import org.zkoss.web.Attributes;

public class MyLocaleProvider implements
org.zkoss.zk.ui.util.RequestInterceptor {
    public void request(org.zkoss.zk.ui.Session sess,
Object request, Object response) {
        final Cookie[] cookies =
((HttpServletRequest)request).getCookies();
        if (cookies != null) {
            for (int j = cookies.length; --j >= 0;) {
```

```
        if (cookies[j].getName().equals("my.locale")) {
            //determine the locale
            String val = cookies[j].getValue();
            Locale locale =
org.zkoss.util.Locales.getLocale(val);
                sess.setAttribute(Attributes.PREFERRED_LOCALE,
locale);
            return;
        }
    }
}
}
```

To make it effective, you have to register it in WEB-INF/zk.xml as a listener. Once registered, the request method is called each time ZK receives a request.

```
<listener>
    <listener-class>MyLocaleProvider</listener-class>
</listener>
```

Note: An instance of the interceptor is instantiated when it is registered. It is then shared among all requests in the same application. Thus, you have to make sure it can be accessed concurrently (i.e., thread-safe).

Note: The `request` method is called at very early stage, before the request parameters are parsed. Thus, it is recommended to access them in this method, unless you configured the locale and character encoding properly for the request.

Change Locale at Run-time

When changing the locale dynamically at the run-time (i.e., under an AU request), it is important to notice:

1. The Locale-dependent messages have been sent to the client, and they have to be reloaded.
2. The current thread's default locale has to be changed since Locale-dependent components and functionality depend on it.

Reload with `sendRedirect`

The simplest way to solve the issues is to ask the browser to reload the whole page by use of `Executions.sendRedirect(java.lang.String)`^[1].

For example,

```
session.setAttribute(Attributes.PREFERRED_LOCALE, locale);
Executions.sendRedirect(null); //reload the same page
```

Notice that `Executions.sendRedirect(java.lang.String)`^[1] will cause the client to reload the page, so any updates to the current desktop will be lost.

Change without Reloading

If you prefer to keep the current desktop, you have to ask the browser to reload the messages, and change the default locale used by the current thread if you're going to access any component and functionality that depends on it. The reloading of messages can be done by invoking Clients.reloadMessages(java.util.Locale)^[8], while the setting of the default locale can be done by the use of Locales.setThreadLocal(java.util.Locale)^[9]

For example,

```
session.setAttribute(Attributes.PREFERRED_LOCALE, locale);
Clients.reloadMessages(locale);
Locales.setThreadLocal(locale);
```

References

- [1] <https://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/Locales.html#getCurrent->
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Application.html#>
- [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/lang/Library.html#>
- [4] <https://support.google.com/chrome/answer/173424?hl=en&co=GENIE.Platform%3DDesktop>
- [5] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/lang/Library.html#setProperty\(java.lang.String,java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/lang/Library.html#setProperty(java.lang.String,java.lang.String))
- [6] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/web/Attributes.html#PREFERRED_LOCALE
- [7] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/RequestInterceptor.html#>
- [8] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#reloadMessages\(java.util.Locale\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#reloadMessages(java.util.Locale))
- [9] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/Locales.html#setThreadLocal\(java.util.Locale\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/Locales.html#setThreadLocal(java.util.Locale))

Time Zone

Overview

The time zone used to process requests and events is, by default, determined by the JVM's default (i.e., java.util.TimeZone.getDefault())^[1].

In this section, we will discuss how to configure ZK to use a time zone other than JVM's default. For example, you might configure ZK to use the preferred time zone that a user specified in his or her profile.

The Decision Sequence of Server Time Zone

The time zone is decided in the following sequence.

1. It checks if an attribute called org.zkoss.web.preferred.timezone defined in the HTTP session (aka., Session^[2]). If so, use it.
2. It checks if an attribute called org.zkoss.web.preferred.timezone defined in the Servlet context (aka., Application^[2]). If so, use it.
3. It checks if a property called org.zkoss.web.preferred.timezone defined in the library property (i.e., Library^[3]). If so, use it.
4. If none of them is found, JVM's default will be used.

You can enforce the time zone with JVM option: -Duser.timezone="Asia/Taipei"

With this sequence in mind, you could configure ZK to use the correct time zone based on the application requirements.

Application-level Time Zone

If you want to use the same time zone for all users of the same application, you can specify the time zone in the library property. For example, you could specify the following in WEB-INF/zk.xml:

```
<library-property>
    <name>org.zkoss.web.preferred.timeZone</name>
    <value>GMT-8</value>
</library-property>
```

- Line 3: the value can be anything accepted by `java.util.TimeZone.getTimeZone()`

Alternatively, if you prefer to specify it in Java, you can invoke `java.lang.String` `Library.setProperty(java.lang.String, java.lang.String)`^[5]. Furthermore, to avoid typos, you could use `Attributes.PREFERRED_TIME_ZONE`^[2] as follows.

```
Library.setProperty(Attributes.PREFERRED_TIME_ZONE, "PST");
```

Per-user Time Zone

Because ZK will check if a session attribute is for the default time zone, you could configure ZK to have per-user time zone by specifying the attribute in a session.

For example, you can do this when a user logins.

```
void login(String username, String password) {
    //check password
    ...
    TimeZone preferredTimeZone = ...; //decide the time zone (from,
say, database)
    session.setAttribute(Attributes.PREFERRED_TIME_ZONE,
preferredTimeZone);
    ...
}
```

Current Time Zone

`TimeZones.getCurrent()`^[3] returns the current (server) time zone determined by the sequence mentioned above.

The Request Interceptor

Like configuring a locale, you can prepare the time zone for the given session by the use of the request interceptor. Please refer to the Locale section for more information.

References

- [1] [https://docs.oracle.com/javase/7/docs/api/java/util/TimeZone.html#getDefault\(\)](https://docs.oracle.com/javase/7/docs/api/java/util/TimeZone.html#getDefault())
- [2] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/web/Attributes.html#PREFERRED_TIME_ZONE
- [3] <https://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/TimeZones.html#getCurrent-->

Handling Server and User Time Zones

Overview

Time is a complicated concept in a web application that serves users across multiple time zones, and especially when serving users in a different time zone than the server itself.

Time and frame of reference

One important point to define when referring to a position in time is "to which point of reference is this time relative?".

In a face-to-face conversation, you could say "I'll see you tomorrow at 10 o'clock in the morning". Since you are both in the same time zone, the assumption would be that you are referring to time relative to your local time zone.

However, if you are calling a person a few time zones away from you, you would need to make the distinction clear. "I'll call you tomorrow at 10 o'clock *my time*", "I'll call you at 10 o'clock *your time*" or even I'll call you at 10 o'clock GMT+0" are usable sentences since they point to a frame of reference for time.

Be sure you know how ZK determines its server time zone by reading Internationalization/Time Zone.

Use of server's time zone

When a client-side date or time value is sent to the ZK server, it is sent as a "moment in global time", which is then automatically converted to the server's local time zone. For example, if a user selected Jan 1st 2022, with a time of 08:00 at GMT+4, the server will receive the matching point in time in its own time zone. If the server is on GMT+0 time zone, for example, it will receive Jan 1st 2022, with a time of 04:00 at GMT+0



Local displayed time	00:00	04:00	08:00
GMT+0 time	00:00	00:00	00:00

Chatbox		Chatbox		Chatbox	
00:01 - User1	Hi, how are you?	04:01 - User1	Hi, how are you?	08:01 - User1	Hi, how are you?
00:03 - User2	Hello! good and you?	04:03 - User2	Hello! good and you?	08:03 - User2	Hello! good and you?
00:03 - User1	Good too!	04:03 - User1	Good too!	08:03 - User1	Good too!

In this case, the displayed times of "GMT+0 00:00", "GMT+4 04:00" and "GMT+8 08:00" all represent the same point in time. They all convert to the same "GMT+0 00:00".

This is necessary if multiple users located in multiple time zones are interacting on the same timeline. For a chat tool, you need to display the time of each message relative to their position on the global shared timeline. Even if that value is converted back to a localized time in the user time zone for display, these timestamps should exist in the same frame of reference from the server's point of view.

Use of client's time zone

Conversely, you may need to use a ZK component that provides a Date and time selection to retrieve a point in time in the user's own local time zone. This could be done to fill a date on a form or to prefill a time limit in a report. In these cases, the important information is the date and time expressed relative to the end-user.

Default time of the day selection in ZK Datebox

The Datebox has the ability to provide date selection with or without time-of-day.

Depending on how the Datebox is set up, it can cater to a number of use cases.

This table shows which position in time-of-day (Hours, minutes, seconds) will be sent to the server as part of the Date selection.

```
//does not contain time information
userInputDatebox.setFormat("yyyy-MM-dd");

//contains time information
userInputDatebox.setFormat("yyyy-MM-dd HH:mm:ss");

//time information was set by server as the time of day and timezone of
the date object passed to the component
java.util.Date existingDateObject = ...;
userInputDatebox.setValue(existingDateObject);

//time information was not set by the server
userInputDatebox.setValue(null); //default if nothing else was set as
value.
```

These choices combine as follow:

	Date set on component by the server	Datebox empty before user selection
Format contains time	Send a date at the selected time of day in the user's time zone, on the selected day	Send a date at the current time of day in the user's time zone on the selected day
Format doesn't contain time	Send a date at the time of day of the server's initial date, on the selected day	Send a date at midnight in the user's time zone on the selected day

Common issue with client time

Based on the table in the previous section, some Date parsing issues may occur if the Datebox was empty before the user's selection. Since an empty Datebox will send back a point in time at midnight in the user's time zone if the user's time zone and the server's time zone are different, the resulting moment in time at the server may be a match "in universal time", but on another day than the selection done "in the user's time zone" due to time zone differences.

A user in GMT-8 may have selected "GMT-8 January 31st at 23:00". If the server time zone is set to GMT+0, the date will be converted to the identical point in time in GMT+0, which is "GMT+0 February 1st at 07:00". If you are using this time to extract a "day of the week", you will retrieve the value a day late.

Inversely, if a user on GMT+8 selects "GMT+8 February 1st at 00:00", the server on GMT+0 will resolve this point in time to "GMT+0 January 31st at 16:00" If you are using this time to extra a "day of the week", you will retrieve a value a day early.

Solutions for client time

There are two ways to resolve the difference between client time zone and server time zone.

Converting server time to client time

Sometimes, it is necessary to display a moment in time to the user. This can be done by showing the full Date and Time information, including the time zone, or by converting the moment in time into a "local display string" matching the user's locale and time zone information.

ZK itself doesn't perform time calculations. As a Java framework, ZK will delegate the task of manipulating time to the relevant Java APIs.

ZK can retrieve the client's locale and time zone information from the `clientInfoEvent`.

A common option is to use Java `DateFormat` to parse and display time in a locale-sensitive manner. For most use cases, `SimpleDateFormat`^[1] is a good option to transform a date object formatted as server time into the user's own time zone.

```
SimpleDateFormat userSimpleDateFormat = new  
SimpleDateFormat("yyyy-MM-dd HH:mm:ss"); //creates a SimpleDateFormat  
formatter  
userSimpleDateFormat.setTimeZone(TimeZone.getTimeZone(clientZoneId));  
//Select the time zone in which the Date object should be displayed  
String formattedTimeInUserTimezone =  
userSimpleDateFormat.format(userInputDatebox.getValue()); //retrieve  
and parse a Date object
```

The resulting string is a representation of the user selected point in time, converted to server Date, and expressed back into the user's own time zone.

Retrieving a `LocalDateTime` object from user input

Starting in ZK 9, ZK Components extending `DateTimeFormatInputElement`^[2] (`Datebox`, `Timebox` and `Timepicker`) provide the option to retrieve the user input value as:

- `LocalDate`^[3] with `DateTimeFormatInputElement#getValueInLocalDate`^[4]
- `LocalTime`^[5] with `DateTimeFormatInputElement#getValueInLocalTime`^[6]
- `LocalDateTime`^[7] with `DateTimeFormatInputElement#getValueInLocalDateTime`^[8]
- `ZonedDateTime`^[9] with `DateTimeFormatInputElement#getValueInZonedDateTime`^[10]

Important note: These values should not be considered reliable options to compare two points in time. Of these 4 options, only `ZonedDateTime` can be semi-reliably converted to a point in time in the server's frame of reference for time. Even so, it would still be possible for errors to happen due to changes in time zone settings, DST, historical changes in time, etc.

These are colloquial expressions of a time in relation to the user's own time exclusively.

They are a convenient way to retrieve a date expressed relative to the user. Any operation that requires date and time conversion should use the previous option and convert server time to client time instead.

Runnable samples

A runnable time and date sample is available in github^[11]. Main composer class^[12] and display zul page^[13].

References

- [1] <https://docs.oracle.com/javase/10/docs/api/java/text/SimpleDateFormat.html>
- [2] <https://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/impl/DateTimeFormatInputElement.html>
- [3] <https://docs.oracle.com/javase/8/docs/api/java/time/LocalDate.html>
- [4] <https://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/impl/DateTimeFormatInputElement.html#getValueInLocalDate-->
- [5] <https://docs.oracle.com/javase/8/docs/api/java/time/LocalTime.html>
- [6] <https://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/impl/DateTimeFormatInputElement.html#getValueInLocalTime-->
- [7] <https://docs.oracle.com/javase/8/docs/api/java/time/LocalDateTime.html>
- [8] <https://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/impl/DateTimeFormatInputElement.html#getValueInLocalDateTime-->
- [9] <https://docs.oracle.com/javase/8/docs/api/java/time/ZonedDateTime.html>
- [10] <https://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/impl/DateTimeFormatInputElement.html#getValueInZonedDateTime-->
- [11] <https://github.com/zkoss/zkbooks/tree/master/developersreference>
- [12] <https://github.com/zkoss/zkbooks/blob/master/developersreference/developersreference/src/main/java/org/zkoss/reference/developer/internationalization/DateboxTimezoneComposer.java>
- [13] <https://github.com/zkoss/zkbooks/blob/master/developersreference/developersreference/src/main/webapp/internationalization/datebox-timezone.zul>

Labels

Overview

For a multilingual application, it is common to display the content in the language that the end user prefers. Here we discuss the built-in support called *internationalization labels*.

However, if you prefer to use other approaches, please refer to the Use Other Implementation section.

Creating Internationalization Labels

The internationalization labels of an application are loaded from properties files based on the current locale^[1]. A properties file is a simple text file encoded in UTF-8^[2]. The file contains a list of key=value pairs, such as^[3]

```
# This is the default LabelsBundle.properties file
s1=computer
s2=disk
s3=monitor
s4=keyboard
```

By default the property file must be placed under the WEB-INF directory and named as zk-label_lang_CNTY.properties.^[4], where lang is the language such as en and fr, and CNTY is the country, such as US and FR.

If you want to use one file to represent a language regardless of the country, you could name it zk-label_lang.properties, such as zk-label_ja.properties. Furthermore, zk-label.properties is the default file if the user's preferred locale doesn't match any other file.

When a user accesses a page, ZK will load the properties files for the user's locale. For example, assume the locale is de_DE, then it will search the following files and load them if found:

1. zk-label_de_DE.properties
2. zk-label_de.properties
3. zk-label.properties

By default, one properties file is used to contain all labels of a given locale. If you prefer to split it into multiple properties files (such as one file per module), please refer to the Loading Labels from Multiple Resources section.

Also, notice that all files that match the given locale will be loaded and merged, and the property specified in, say, `zk-label_de_DE.properties` will override what is defined in `zk-label_de.properties` if replicated. It also means if a label is the same in both `de_DE` and `de`, then you need only to specify in `zk-label_de.properties` (and then it will be *inherited* when `de_DE` is used). Of course, you could specify it in both files.

[1] It is the value returned by UNIQ-javadoc-0-736dd3f9533a2ca2-QINU . For more information, please refer to the Locale section.

[2] If you prefer a different charset, please refer to the Encoding Character Set section.

[3] Please refer to here for more details about the format of a properties file, such as the use of multiple lines and EL expressions.

[4] Notice the directory and filename are configurable. For more information, please refer to ZK Configuration Reference:
`org.zkoss.util.label.web.location`

Encoding character set

By default, the encoding of properties files is assumed to be UTF-8. If you prefer another encoding, please specify it in a library property called `org.zkoss.util.label.web.charset`. It also means all properties files must be encoded in the same character set.

For more information, please refer to ZK Configuration Reference.

Access Internationalization Labels In ZUML

Use labels

Since 5.0.7 and later, an implicit object called `labels` was introduced, such that you could access the internationalization labels (so-called internationalization labels) directly. For example, assume you have a label called `app.title`, and then you could:

```
<window title="${labels.app.title}">
...
</window>
```

The `labels` object is a map (`java.util.Map`), so you could access the label directly by the use of `labels.whatever` in an EL expression. Moreover, as shown above, you could access the label even if a key is named as `aa.bb.cc` (a string containing dots), such as `app.title` in the above example.

If the key is not a legal name, you could use `labels['key']` to access it, such as `labels['foo-yet']`.

When an internationalization label is about to be retrieved, one of `zk-label_lang_CNTY.properties` will be loaded. For example, if the Locale is `de_DE`, then `WEB-INF/zk-label_de_DE.properties` will be loaded. If no such file, ZK will try to load `WEB-INF/zk-label_de.properties` and `WEB-INF/zk-label.properties` in turn.

Notice that ZK groups the segmented labels as maps. For example, `${labels.app}` was resolved as a map containing two entries (`title` and `description`).

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

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

For example, in properties file:

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

In ZUL:

```
<window title="${labels.app.$}"><!-- shows "Application" -->  
...  
</window>  
<window title="${labels.app}"><!-- WRONG! -->  
...  
</window>
```

Use c:l('key')

With 5.0.6 or prior, you could get an internationalization label using \${c:l('key')} in EL expression. For example,

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

Notice that the l function belongs to the TLD file called http://www.zkoss.org/dsp/web/core, so we have to specify it with the taglib directive as shown above.

Use c:l2('key') to format the message

If you'd like to use the label as a pattern to generate concatenated message with additional arguments (like java.text.MessageFormat (<https://docs.oracle.com/javase/8/docs/api/java/text/MessageFormat.html>) does), you could use the l2 function.

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

```
fullname.format=full name is {0}
```

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

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

`java.lang.Object[(http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/resource/Labels.html#getLabel(java.lang.String,)) Labels.getLabel(java.lang.String, java.lang.Object[])]` assumes the content is a valid pattern accepted by `MessageFormat` (<http://download.oracle.com/javase/6/docs/api/java/text/MessageFormat.html>), such as "`{1}, {0}`".

Please notice that "a single quote itself must be represented by doubled single quotes" according to `java.text.MessageFormat` (<https://docs.oracle.com/javase/8/docs/api/java/text/MessageFormat.html>).

Access Internationalization Labels In Java

To access labels in Java code (including zscript), you could use `Labels.getLabel(java.lang.String)` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/resource/Labels.html#getLabel\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/resource/Labels.html#getLabel(java.lang.String))), `java.lang.Object[(http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/resource/Labels.html#getLabel(java.lang.String,)) Labels.getLabel(java.lang.String, java.lang.Object[])]` and others.

```
String username = Labels.getLabel("username");
```

Here is a more complex example. Let us assume we want to generate a full name based on the Locale, then we could use `java.lang.Object[(http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/resource/Labels.html#getLabel(java.lang.String,)) Labels.getLabel(java.lang.String, java.lang.Object[])]` to generate concatenated messages as follows.

```
public String getFullName(String firstName, String lastName) {
    return Labels.getLabel("fullname.format", new java.lang.Object []
{firstName, lastName});
}
```

`java.lang.Object[(http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/resource/Labels.html#getLabel(java.lang.String,)) Labels.getLabel(java.lang.String, java.lang.Object[])]` assumes the content is a valid pattern accepted by `MessageFormat` (<http://download.oracle.com/javase/6/docs/api/java/text/MessageFormat.html>), such as "`{1}, {0}`".

Loading Labels from Multiple Resources

It is typical to partition the properties file into several modules for easy maintenance. Since 5.0.7 and later, you could specify the location for each of these properties file with the `label-location` element. For example,

```
<system-config>
    <label-location>/WEB-INF/labels/order.properties</label-location>
    <label-location>/WEB-INF/labels/invoice.properties</label-location>
</system-config>
```

Notice that, once you specify `label-location`, the default loading of `/WEB-INF/zk-labels.properties` won't take place. In other words, only the properties files specified in the `label-location` elements are loaded. Thus, if you'd like to load `/WEB-INF/zk-labels.properties` too, you have to add it to `label-location` with others.

Also notice that you don't have to and shall not specify the language, such as `de_DE`, in the path. ZK will try to locate the most matched one as described in the previous section.

In addition to the servlet path, you could specify a file path by starting with `file://`^[1]. For example, `file:///foo/labels.properties`. If the target environment is Windows, you could specify the drive too,

such as `file:///C:/myapp/foo.properties`. The advantage is that additional properties files could be added after the project has been built into a WAR file.

```
<system-config>
    <label-location>file:///labels/order.properties</label-location>
    <label-location>file:///labels/invoice.properties</label-location>
</system-config>
```

Notice that the configuration with a path related to the file system is better not to be part of `WEB-INF/zk.xml`, since it is easy to cause errors when deploying the application. Rather, it is better to be specified in the additional configuration file. The additional configuration file is also specified at the run time and could be located in the file system (rather than the WAR file). It can be done by specifying the path of the configuration file in a library property called `org.zkoss.zk.config.path`.

For 5.0.6 and older, you could use the approach described in the following section to load multiple properties files.

[1] For more information about the URI of a file, please refer to File URI scheme (http://en.wikipedia.org/wiki/File_URI_scheme).

Loading Labels from Jar

If your application is built using multiple Jars as custom components or as a modular project, you can load internationalization labels by putting the .properties files in the resource folder of your add-on project.

Required Steps:

1. put properties files under [classpath]/metainfo
2. the properties files name should be `zk-label.properties` or `zk-label_[LOCALE].properties`
e.g. `zk-label_en.properties`

For example, if you are building with maven, the files can be placed into `/src/main/resources/metainfo/` in your project. You can define the default labels using `zk-label.properties` as well as language specific labels using the same file name convention as in the default case.

When the jars generated this way are added as libraries to the main ZK project, the properties files located in these libraries will be used to locate labels as well as the properties files declared in the main application. The properties files must follow the same syntax used in the default case.

Loading from Database or Other Resources

If you prefer to put the internationalization labels in, say, database, you could extend the label loader to load labels from other locations, say database. It can be done by registering a locator, which must implement either `LabelLocator` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/resource/LabelLocator.html#>) or `LabelLocator2` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/resource/LabelLocator2.html#>). Then, invoking `Labels.register(org.zkoss.util.resource.LabelLocator)` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/resource/Labels.html#register\(org.zkoss.util.resource.LabelLocator\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/resource/Labels.html#register(org.zkoss.util.resource.LabelLocator))) or `Labels.register(org.zkoss.util.resource.LabelLocator2)` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/resource/Labels.html#register\(org.zkoss.util.resource.LabelLocator2\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/resource/Labels.html#register(org.zkoss.util.resource.LabelLocator2))) to register it^[1].

If you can represent your resource in URL, you could use `LabelLocator` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/resource/LabelLocator.html#>) (as shown below). If you have to load it by yourself, you could use `LabelLocator2` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/resource/LabelLocator2.html#>) and return an input stream (`java.io.InputStream`).

Alternative 1: load as an input stream:

```
public class FooDBLocator implements
org.zkoss.util.resource.LabelLocator2 {
    private String _field;
    public FooDBLocator(String field) {
        _field = field;
    }
    public InputStream locate(Locale locale) {
        InputStream is = ... //load the properties from, say, database
        return is;
    }
    public String getCharset() {
        return "UTF-8"; //depending the encoding you use
    }
}
```

Alternative 2: load as an URL:

```
public class FooServletLocator implements
org.zkoss.util.resource.LabelLocator {
    private ServletContext _svlctx;
    private String _name;
    public FooServletLocator(ServletContext svlctx, String name) {
        _svlctx = svlctx;
        _name = name;
    }
    public URL locate(Locale locale) {
        return _svlctx.getResource("/WEB-INF/labels/" + name + "_" +
locale + ".properties");
    }
}
```

Then, we could register label locators when the application starts by use of WebAppInit (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/WebAppInit.html#>) as follows.

```
public class MyAppInit implements org.zkoss.zk.ui.util.WebAppInit {
    public void init(WebApp wapp) throws Exception {
        Labels.register(new FooDBLocator("moduleX"));
        Labels.register(new FooDBLocator("moduleY"));
        Labels.register(new
FooServletLocator((ServletContext)wapp.getNativeContext(), "module-1"));
        Labels.register(new
FooServletLocator((ServletContext)wapp.getNativeContext(), "module-2"));
    }
}
```

where we assume moduleX and moduleY are the database tables to load the properties, and module-1.properties and module-2.properties are two modules of messages you provide. Then, you configure it in WEB-INF/zk.xml as described in ZK Configuration Reference.

[1] For 5.0.7 and later, you could use the label-location element if the properties file is located in the file system or in the Web application as described in the previous section.

Reload Labels Dynamically

The internationalization labels are loaded when a locale is used for the first time. It won't be reloaded automatically if the file is modified. However, it is easy to force ZK to reload by the use of Labels.reset() ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/resource/Labels.html#reset\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/resource/Labels.html#reset())).

For example, you could prepare a test paging for reloading as follows.

```
<zk>
<button label="Reload Labels" onClick="org.zkoss.util.resource.Labels.reset();execution.sendRedirect(null);"/>
Test result: ${foo} ${another.whatever}
</zk>
```

Use Other Implementation

If you prefer to use other implementation (such as property bundle), you could implement a static method and map it with xel-method. Then, you could reference it in EL expressions. For example,

```
<?xel-method prefix="c" name="label" class="foo.MyI18Ns"
  signature="java.lang.String label(java.lang.String)"?>
<window title="#{c:label('app.title')}">
  ...
  ${c:label('another.key')}
</window>
```

Version History

Version	Date	Content
5.0.5	October 2010	LabelLocator2 (http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/resource/LabelLocator2.html#) was introduced.
5.0.7	March 2011	The <code>labels</code> object was introduced.

The Format of Properties Files

In this section, we will discuss the format of a properties file, such as `zk-label.properties`.

A properties file is a simple text file. The file contains a list of `key=value` pairs, such as

```
# This is the default LabelsBundle.properties file
s1=computer
s2=disk
s3=monitor
s4=keyboard
```

The default encoding of a properties file is assumed to be **UTF-8**. If you want to use a different encoding, please refer to the Use Encoding Other Than UTF-8 section.

A properties file is usually used to contain the internationalization labels of an application, but technically you could use it in any situation you'd like^[1].

[1] If it is used for internationalization labels, it will be loaded automatically. If you want to use it in other situations, you have to invoke UNIQ-javadoc-0-736dd3f9533a2ca2-QINU or similar to load it manually.

Specify a Value with Multiple Lines

By default, a property is a text specified right after the equal sign. If the property's value has multiple lines, you could use the following format:

```
multilines={
line 1
line 2
}
```

Notice that the curly braces must be followed by a line break immediately, and the right brace () must be the only character in the line.

Then you should put the value in a multiple-line label:

```
<label multiline="true" value="${labels.multilines}" />
```

Render Multiple Lines

Alternatively, you can write a value with `
`:

```
lines=1st line <br> 2nd line
```

Then render it in an HTML span

```
<zk xmlns:h="native">
  <h:span>${labels.lines}</h:span>
</zk>
```

Specify Segmented Keys

Since all internationalization labels are stored in the same scope, it is common to separate them by naming the key with dots (.) like the Java package name. For the sake of description, we call them segmented keys. For example,

```
order.fruit.name = Orange
order.fruit.description = A common fruit
```

It can be simplified by use of the following syntax:

```
order.fruit. {
    name = Orange
    description = A common fruit
}
```

As shown, the segmented key could be specified by specifying the prefix and a following right brace ({}).

The segmented key could be accessed in two ways.

First, with an implicit object called labels:

```
<textbox value="${labels.order.fruit.name}" />
```

Second, with an EL function called l and/or l2:

```
<?taglib uri="http://www.zkoss.org/dsp/web/core" prefix="c"?>
<label value="${c:l('order.fruit.name')}">
```

Under the hood

The labels object is actually the map returned by Labels.getSegmentedLabels() (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/resource/Labels.html#%24getSegmentedLabels%28%29>). Furthermore, if the key of a property contains dots (.), i.e., segmented, all properties with the same prefix are grouped as another map. For example, \${labels.order} (i.e., Labels.getSegmentedLabels().get("order")) will return a map containing an entry (fruit) in the above example.

Specify a Comment

You could put a comment line by starting with the sharp sign (#), such as

```
#This is a comment line that will be ignored when loaded
```

Use EL Expressions

EL expressions are allowed for a property's value. For example, you could reference a property's value in another property, such as

```
first=the first label
second=come after ${first}
```

Segmented keys are also allowed:

```
group1.first=the first group
group2.second=come after ${group1.first}
```

In addition to referencing another property, you could reference any implicit object specified in ZUML Reference: Implicit Objects if it is part of an HTTP request (excluding component/page).

For example, param references to a request's parameter:

```
message=Thank ${param.user} for using
```

Use Encoding Other Than UTF-8

By default, the encoding of properties files is assumed to be UTF-8. If you prefer another encoding, please specify it in a library property called `org.zkoss.util.label.web.charset`. It also means all properties files must be encoded in the same character set.

For more information, please refer to ZK Configuration Reference.

Version History

Version	Date	Content
5.0.7	Mar 2011	labels implicit object was introduced to access properties without declaring taglib. Also allows label keys of a.b.c format.

Date and Time Formatting

Overview

By default, the format of date and time, especially the format of Datebox and Timebox, is determined by the JVM's default and the current locale.

In this section, we will discuss how to configure ZK to use a format other than the JVM. For example, you could configure ZK to use the preferred format based on the user's preferences.

The Decision Sequence of Format

The format of date and time is decided in the following sequence.

1. It checks if an attribute called `org.zkoss.web.preferred.dateFormatInfo` is defined in the HTTP session (i.e., Session^[2]). If so, it will be used by assuming the value is an instance or a class of `DateFormatInfo`^[1].
2. It checks if an attribute called `org.zkoss.web.preferred.dateFormatInfo` is defined in the servlet context (i.e., Application^[2]). If so, it will be used by assuming the value is an instance or a class of `DateFormatInfo`^[1].
3. It checks if a property called `org.zkoss.web.preferred.dateFormatInfo` is defined in the library property (i.e., Library^[3]). If so, it will be used by assuming the value is a class of `DateFormatInfo`^[1].
4. If none of them is found, it uses the JVM's default based on the current locale.

In other words, to configure ZK to use a format other than the JVM's default, you have to:

1. Implements `DateFormatInfo`^[1] to provide the format you want.
2. Specify the class or an instance of it in the session's attribute or application's attribute depending on the requirement of your application.

If a class or the class's name is specified, an instance of it is instantiated each time the server receives a request from the client. It means the implementation needs not to be thread safe, but at the cost of instantiation. On the other hand, if an instance is specified as the attribute value, it will be used for all requests, so it has to be thread safe.

Also notice that you could specify `short`, `long` and other *standard* styling in the `format` property of datebox and timebox, such that the corresponding format of the styling will be used instead of the default, `medium`. For more information, please refer to the Per-component Format section.

Application-level Format

If you want to use the same format for all users, you could specify your implementation of `DateFormatInfo`^[1] in the `library` property. For example,

```
<library-property>
  <name>org.zkoss.web.preferred.dateFormatInfo</name>
  <value>foo.MyDateFormatInfo</value>
</library-property>
```

where we assume the implementation is named `foo.MyDateFormatInfo`.

Per-user Format

If you'd like to configure ZK to allow each user (aka., session) to have an independent format, you could store an instance of your implementation of `DateFormatInfo`^[1] in the session's attribute.

For example, you could do this when a user logs in.

```
import org.zkoss.web.Attributes;
...
void login(String username, String password) {
    //check password
    ...
    session.setAttribute(Attributes.PREFERRED_DATE_FORMAT_INFO,
        new foo.MyDateFormatInfo(session));
    ...
}
```

where we assume the implementation is named `foo.MyDateFormatInfo`.

Per-component Format

Datebox and Timebox allow a developer to specify any format they prefer for any instance. For example,

```
<datebox format="MM d, yyyy"/>
<timebox format="HH:mm"/>
```

However, it is usually better to design a page that depends on the configuration as described above, rather than specify the format explicitly in each page. It can be done by specifying the styling rather than the real format in the `format` property (`Datebox.setFormat(java.lang.String)`^[2] and `Timebox.setFormat(java.lang.String)`^[3]). There are four different styles: `short`, `medium`, `long` and `full` (representing the styling defined in `java.text.DateFormat`, `SHORT`, `MEDIUM`, `LONG` and `FULL`). For example,

```
<datebox format="short"/>
<datebox format="long"/>
<timebox format="medium"/>
```

Then, the real format will be decided by your implementation of DateFormatInfo^[1], if any, or the JVM's default.

In addition, you could specify the date/time format in the syntax of styling_for_date+styling_for_time, such as:

```
<datebox format="long+medium"/>
```

which specifies the date/time format with the long styling for date and the medium styling for time.

Per-component Locale

In addition to the current locale, you could specify the locale for individual instances of datebox and timebox. Then, the real format will depend on the locale and the format you specified. For example,

```
<datebox format="medium" locale="de"/>
<timebox format="long" locale="fr"/>
```

Note: the language of the format and the datebox's calendar is the same as the locale you specified.

Customization

ZK generates all date related strings in Java according to locale e.g. the day of the week (Sun, Mon, Tue...). If you need a customized format or a locale that JDK doesn't support. You can create such date related text by yourselves.

The default values in English are:

```
zk.DOW_1ST=0;
zk.MINDAYS=1;
zk.ERA="AD";
zk.YDELTA=0;
zk.SDOW=['Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat'];
zk.S2DOW=zk.SDOW;
zk.FDOW=['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday'];
zk.SMON=['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'];
zk.S2MON=zk.SMOM;
zk.FMON=['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November', 'December'];
zk.APM=['AM', 'PM'];
```

You can override just part of them like:

```
zk.afterLoad('zul.lang', function() {
    zk.SDOW=['Sun_gl', 'Mon_gl', 'Tue_gl', 'Wed_gl', 'Thr_gl', 'Fri_gl', 'Sat_gl'];
    //set your date string
}); //zk.afterLoad
```

You can load the JavaScript file as a locale dependent JavaScript to override the date string for a specific locale.

Version History

Version	Date	Content
5.0.7	April 2011	The per-session format of datebox/timebox was introduced. Prior to 5.0.7, the format depends only on locale.

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/text/DateFormatInfo.html#>
- [2] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Datebox.html#setFormat\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Datebox.html#setFormat(java.lang.String))
- [3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Timebox.html#setFormat\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Timebox.html#setFormat(java.lang.String))

The First Day of the Week

Overview

By default, the first day of the week depends on the locale (e.g., Sunday in US, Monday in France). More precisely, it is the value returned by the `getFirstDayOfWeek` method of the `java.util.Calendar` class.

However, you can configure it differently, and it will affect how datebox and calendar components behave.

The decision sequence of the first day of the week

The first day of the week is decided in the following sequence.

1. It checks if an attribute called `org.zkoss.web.preferred.firstDayOfWeek` is defined in the HTTP session (aka., Session^[2]). If so, use it.
2. It checks if an attribute called `org.zkoss.web.preferred.firstDayOfWeek` is defined in the Servlet context (aka., Application^[2]). If so, use it.
3. It checks if a property called `org.zkoss.web.preferred.firstDayOfWeek` is defined in the library property (i.e., Library^[3]). If so, use it.
4. If none of them is found, JVM's default will be used (`java.util.Calendar`).

Application-level first-day-of-the-week

If you want to use the same first-day-of-the-week for all users of the same application, you can specify it in the library property. The allowed values include 1 (Sunday), 2 (Monday), .. and 7 (Saturday). For example, you could specify the following in WEB-INF/zk.xml:

```
<library-property>
  <name>org.zkoss.web.preferred.firstDayOfWeek</name>
  <value>7</value><!-- Saturday -->
</library-property>
```

Alternatively, if you prefer to specify it in Java, you could invoke `java.lang.String`) `Library.setProperty(java.lang.String, java.lang.String)`^[5]. Furthermore, to avoid typos, you could use `java.lang.Object` `WebApp.setAttribute(java.lang.String, java.lang.Object)`^[1] and `Attributes.PREFERRED_FIRST_DAY_OF_WEEK`^[2] as follows.

```
webApp.setAttribute(org.zkoss.web.Attributes.PREFERRED_FIRST_DAY_OF_WEEK,  
java.util.Calendar.SATURDAY);
```

As shown above, the allowed values of java.lang.Object) WebApp.setAttribute(java.lang.String, java.lang.Object)^[1] include Calendar.SUNDAY, Calendar.MONDAY and so on.

Per-user first-day-of-week

By specifying a value to the session attribute called org.zkoss.web.preferred.firstDayOfWeek (i.e., Attributes.PREFERRED_FIRST_DAY_OF_WEEK^[2]), you can control the first day of the week for the given session. The allowed values include Calendar.SUNDAY, Calendar.MONDAY and so on.

```
session.setAttribute(org.zkoss.web.Attributes.PREFERRED_FIRST_DAY_OF_WEEK,  
java.util.Calendar.SATURDAY);  
//then, the current session's first day of the week will be Saturday
```

For example, you can do this when a user logins.

```
void login(String username, String password) {  
    //check password  
    ...  
    int preferredFDOW = ...; //decide the user's preference  
    session.setAttribute(Attributes.PREFERRED_FIRST_DAY_OF_WEEK,  
preferredFDOW);  
    ...
```

References

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/WebApp.html#setAttribute\(java.lang.String,%20java.lang.Object\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/WebApp.html#setAttribute(java.lang.String,%20java.lang.Object))

[2] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/web/Attributes.html#PREFERRED_FIRST_DAY_OF_WEEK

Locale-Dependent Resources

Overview

Many resources depend on the Locale and, sometimes, the browser. For example, you might need to use a larger font for Chinese characters to have better readability. ZK provides a way to load locale and browser-dependent resources including JavaScript, CSS, and images.

Specifying Locale- and browser-dependent URL

ZK can handle this for you automatically by specifying a URL with **asterisk ***. This feature is supported by all components that accept a URL, e.g. the src of <script> or <?script?>. The algorithm is as follows.

Absolute or Relative Path

You can only specify an absolute path to load a locale-dependent resource at 9.x and before.
Relative path is supported.

One "*" for Locale Code

If you specify only one * in a URL, such as /my*.css, then * will be replaced with a **locale code** depending on the current locale that ZK detects. For example:

- URI: /**my***.css
- User's preferences (current locale): **de_DE**

Then ZK looks for files in the order below one-by-one in your website until any of them is found:

1. /my_de_DE.css
2. /my_de.css
3. /my.css

Two "*" for Locale and Browser Code

Such as "/my*/lang*.css", then the first "*" will be replaced with a **browser code** as follows:

- ie for Internet Explorer
- saf for Chrome, Safari
- moz for Firefox and other browsers^[1].

Moreover, the last asterisk will be replaced with a proper Locale as described in the previous rule. In summary, the last asterisk represents the Locale, while the first asterisk represents the browser type.

For example:

zul

```
<style src="/i18n/css-*/*mycss*.css" />
```

The result in an HTML with Chrome:

```
<link ... href="/i18n/css-saf/mycss.css" ...>
```

Note

The last asterisk that represents the Locale must be placed right before the first dot ("."), or at the end if no dot at all. Furthermore, no following slash (/) is allowed, i.e., it must be part of the filename, rather than a directory. If the last asterisk doesn't fulfill this constraint, it will be eliminated (not ignored).

For example, "/my/lang.css*" is equivalent to "/my/lang.css".

In other words, you can consider it as neutral to the Locale.

Tip: We can apply this rule to specify a URI depending on the browser type, but not depending on the Locale. For example, "/my/lang*.css*" will be replaced with "/my/langie.css" if Internet Explorer is the current user's browser.

[1] In the future editions, we will use different codes for browsers other than Internet Explorer, Firefox and Safari.

Example

In the following example, we assume the preferred Locale is de_DE and the browser is Internet Explorer.

URI	Resources that are searched
/css/norm*.css	1. /norm_de_DE.css 2. /norm_de.css 3. /norm.css
/css-*/norm*.css	1. /css-ie/norm_de_DE.css 2. /css-ie/norm_de.css 3. /css-ie/norm.css
/img*/pic*/lang*.png	1. /imgie/pic*/lang_de_DE.png 2. /imgie/pic*/lang_de.png 3. /imgie/pic*/lang.png
/img*/lang.gif	1. /img/lang.gif
/img/lang*.gif*	1. /img/langie.gif
/img*/lang*.gif*	1. /imgie/lang*.gif

Locating Locale- and browser-dependent resources in Java

In addition to ZUML, you can also load browser- and Locale-dependent resources in Java. Here is a list of methods that you can use.

- The `encodeURL`, `forward`, and `include` methods in Execution (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#>) for encoding URL, forwarding to another page and including a page. In most cases, these methods are all you need.
- The `locate`, `forward`, and `include` method in Servlets (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/web/servlet/Servlets.html#>) for locating Web resources. You rarely need them when developing ZK applications, but useful for writing a servlet, portlet or filter.
- The `encodeURL` method in Encodes (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/web/servlet/http/Encodes.html#>) for encoding URL. You rarely need them when developing ZK applications, but useful for writing a Servlet, Portlet or Filter.
- The `locate` method in Locators (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/resource/Locators.html#>) for locating class resources.

Warning and Error Messages

Overview

ZK's system locale dependent messages (such as warnings and errors) are stored in ZK jar with 2 formats:

1. [ZK-JAR]/metainfo/mesg/*.properties, Java properties files
[ZK-JAR] could be zcommon.jar, zweb.jar, zk.jar or zul.jar.
2. zk.jar /web/js/zk/lang/*.js
3. zul.jar/web/js/zul/lang/*.js

These files are Locale depedent. For example, the message file in zk.jar for German messages are

- /metainfo/mesg/msgzk_de.properties,
- /web/js/zk/lang/msgzk_de.js

Translate messages to another language

If you want to translate messages to another language, you can add your own property files named with the correct Locale, and put them to the /metainfo/mesg directory of the classpath. Of course, it is always better to contribute back. Please take a look at ZK Messages for all available translations. If you'd like to contribute, just add the language to it and notify us at info@zkoss.org.

Change particular message

Look for the specific message you want to change at ZK Messages first. Then, override it according to the following sections.

Defined in a Properties File

If you want to change a particular message, you need to create WEB-INF/zk-label.properties (or WEB-INF/zk-label_[LOCALE].properties) and add key-value pairs in it. For example, assuming you want to customize MZk.NOT_FOUND in German translation (msgzk_de.properties), then you can add the following to WEB-INF/zk-label_de.properties:

```
MZk.3000=my customized message here
```

Notice the prefix MZk, and 3000 is the error code and you can find it at ZK_Messages/German/msgzk_de.properties

The same pattern applies to the other message files such as

File	Prefix of a Key	Key Example
msgzk_[LOCALE].properties	MZk	MZk.1000= my text
msgzul_[LOCALE].properties	MZul	MZul.2400=mytext
msgcommon_[LOCALE].properties	MCommon	MCommon.1234= my text
msgweb_[LOCALE].properties	MWeb	MWeb.1234= my test

Defined in a JS File

Using local override

For messages defined in msgzk.js / msgzul.js you can create a js file and include it via lang-addon.xml. For example for overriding msgzk.LOADING for CEZH language create a file test_cs.js and override particular message msgzk.LOADING=CS langauge specific message" and include this js file via lang-addon.xml as below

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

  <javascript src="/js/test*.js" />
</language-addon>
```

Note: You can use any prefix other than "test" and add language-specific suffixes to your js files and include all of them by using * wild card as shown above

Finally include your lang-addon.xml in zk.xml using language-config element as shown below

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

Note: messages updated by a custom javascript will override the default once. If the messages are reloaded using Clients.reloadMessages(Locale), the customization will be lost. To support locale reload, please refer to the next section.

Adding a custom language loader

A custom language loader will find matching messages depending on the user locale, and will be reloaded even if the Clients.reloadMessages(Locale) is triggered.

It can be defined in a language addon using the <message-loader-class> element. Please refer to the client-side reference documentation for more details.

Version History

Version	Date	Content
6.0.0	n/a	Allows applications to override a particular message with zk-label.

Server Push

HTTP is a request-and-response protocol. Technically, there is no way to have the server to actively *push* data to the client. However, there are a few approaches ^[1] to emulate *push* -- it is also called Ajax Push. We can summarize these approaches in 2 categories:

1. client polling and
2. comet (more precisely, it is the so-called long polling)

They are both supported in ZK.

Different approaches have different pros and cons, and we will discuss them in the Configuration section.

No matter which implementation you choose, the usage is the same. The Event Queue is the high-level API, and this is a suggested approach for its simplicity. However, if you prefer to access the low-level API directly, you could refer to the Asynchronous Tasks and Synchronous Tasks sections, depending on whether your task can be executed asynchronously.

Browser Concurrent Connection Limitation

The server push (exception polling) establishes an open connection. Modern web browsers typically impose a limitation on the number of simultaneous HTTP connections that can be established with a single domain. According to the HTTP specification outlined in RFC2616, this constraint is commonly set to a maximum of **6** concurrent connections per domain.

See Browser connection limitations ^[2].

References

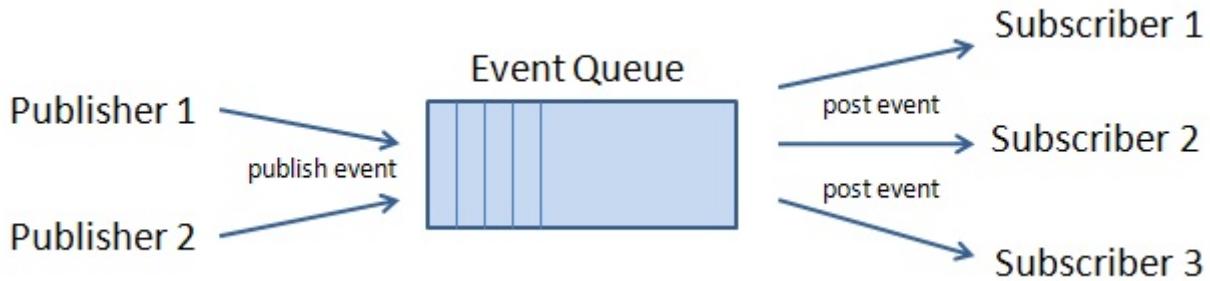
[1] http://en.wikipedia.org/wiki/Push_technology

[2] https://docs.diffusiondata.com/cloud/latest/manual/html/designguide/solution/support/connection_limitations.html

Event Queues

An event queue is an event-based publish-subscribe solution for the application information delivery and messaging. It provides asynchronous communication for different modules/roles in a loosely-coupled and autonomous fashion.

By publishing, a module (publisher) sends out messages without explicitly specifying or having knowledge of intended recipients. By subscribing, a receiving module (subscriber) receives messages that the subscriber has registered an interest in, without explicitly specifying or knowing the publisher.



ZK generalizes the event queue to support the server push. The use is straightforward: specifying the scope of a given event queue as `EventQueues.APPLICATION`^[2] (or `EventQueues.SESSION`^[1], but rare). For example,

```
EventQueue que = EventQueues.lookup("chat", EventQueues.APPLICATION,
true);
```

For more information about event queues, please refer to the Event Handling: Event Queues section.

For the information about low-level API, please refer to Asynchronous Tasks section, if the task can execute asynchronously; or Synchronous Tasks if it must execute synchronously.

Version History

Version	Date	Content
5.0.6	November 2010	The event queue won't start any working threads and they are serializable, so it is safe to use them in a clustering environment.

References

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

Synchronous Tasks

Server push is a technology to actively *push* data to the client. For ZK, the data is usually the UI updates or its variants. Thus, for the sake of understanding, we could consider the task to be about updating UI in parallel with regular Ajax requests (poll-type requests). For example, in a chat application, once a message is entered by a participant, the server has to *push* it to all clients that are involved in the conversation.

If the task of updating UI takes place in a working thread, it is generally more convenient to execute it synchronously as described later. On the other hand, if the task can be encapsulated as an event listener (EventListener^[1]), you could execute it asynchronously -- please refer to the Asynchronous Tasks section for more information.

Enable Server Push

By default, the server push is disabled (for better performance). Before pushing data for a given desktop, you have to enable the server push for it.

It can be done by use of Desktop.enableServerPush(boolean)^[1]:

```
desktop.enableServerPush(true);
```

After the server push of a given desktop is enabled, you could use any number of working threads to update the desktop concurrently as described in the following section^[2].

[1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Desktop.html#enableServerPush\(boolean\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Desktop.html#enableServerPush(boolean))

[2] For better performance, it is suggested to disable the server push if it is no longer used in the given desktop.

Update UI in a Working Thread

To update the UI synchronously in a working thread, we have to do as follows.

1. Invoke Executions.activate(org.zkoss.zk.ui.Desktop) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#activate\(org.zkoss.zk.ui.Desktop\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#activate(org.zkoss.zk.ui.Desktop))). It has two purposes:
 1. It grants the right to access the UI of the given desktop to the caller's thread.
Notice that, for each desktop, at most one thread is allowed to access at the same time.
 2. It establishes a connection with the client (the browser window displaying the desktop), such that the update will be sent to the client after finished.
2. Update UI any way you want, just like any regular event listener.
3. Invoke Executions.deactivate(org.zkoss.zk.ui.Desktop) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#deactivate\(org.zkoss.zk.ui.Desktop\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#deactivate(org.zkoss.zk.ui.Desktop))) to return the control, such that other threads could have a chance to update UI.

Here is an example code that illustrates the usages:

```
public class WorkingThread extends Thread {  
    public void run() {  
        try {  
            while (anyDataToShow()) {  
                //Step 1. Prepare the data that will be updated to UI  
                collectData(); //prepare the data to set to components  
  
                //Step 2. Activate to grant the access of the given  
                desktop
```

```

        Executions.activate(desktop);
    try {
        //Step 3. Update UI
        updateUI(); //implement the logic to change UI,
call ZK component API or notify change
    } finally {
        //Step 4. Deactivate to return the control of UI
back
        Executions.deactivate(desktop);
    }
}

} catch (InterruptedException ex) {
    //Interrupted. You might want to handle it
}
}

/*
 * To update UI you can do one of the followings:
 * - to notify changes with {@link
BindUtils#postNotifyChange(Object, String)} if changing a ViewModel's
property
 * - call a component's setter
 * - If calling a {@link org.zkoss.zul.ListModel} method, it
automatically updates for you without notifying
 */
protected void updateUI() {
    data.clear();
    data.add("now " + System.currentTimeMillis());
}
}

```

- Line 9-15: the task between Executions.activate(org.zkoss.zk.ui.Desktop) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#activate\(org.zkoss.zk.ui.Desktop\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#activate(org.zkoss.zk.ui.Desktop))) and Executions.deactivate(org.zkoss.zk.ui.Desktop) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#deactivate\(org.zkoss.zk.ui.Desktop\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#deactivate(org.zkoss.zk.ui.Desktop))) has to take less time, since it blocks others, including the end users (of the desktop), from accessing the UI. It is suggested to prepare the data before Executions.activate(org.zkoss.zk.ui.Desktop) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#activate\(org.zkoss.zk.ui.Desktop\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#activate(org.zkoss.zk.ui.Desktop))), such that it can be done in parallel with other threads.

For a real example, please refer to small talks: Simple and Intuitive Server Push with a Chat Room Example and Server Push with a Stock Chart Example.

Asynchronous Tasks

If you run an application logic in a task thread (not in a servlet thread), and you don't want to update UI in the same thread. All you need to do is:

1. enable server push
2. Implement the UI updates in an event listener (implement EventListener^[1] or SerializableEventListener^[2]).
3. Execute the listener asynchronously by org.zkoss.zk.ui.event.EventListener, org.zkoss.zk.ui.event.Event) Executions.schedule(org.zkoss.zk.ui.Desktop, org.zkoss.zk.ui.event.EventListener, org.zkoss.zk.ui.event.Event)
[1].

Here is the code snippet:

```
@Listen("onClick = #start")
public void start() throws ExecutionException, InterruptedException
{
    // run in a separate thread
    CompletableFuture.runAsync(() -> {
        Threads.sleep(3000); //simulate a long task
        Executions.schedule(desktop,
            new EventListener<Event>() {
                public void onEvent(Event event) {
                    //update UI
                    status.setValue("done at " +
LocalDateTime.now());
                }
            }, new Event("myEvent"));
    });
}
```

- Line 10: You can manipulate ZK UI components in EventListener.onEvent(org.zkoss.zk.ui.Event)^[2]. It is no different from any other event listener.

Notice that org.zkoss.zk.ui.event.EventListener, org.zkoss.zk.ui.event.Event Executions.schedule(org.zkoss.zk.ui.Desktop, org.zkoss.zk.ui.event.EventListener, org.zkoss.zk.ui.event.Event)^[1] can be called anywhere, including another event listener or a task thread. In other words, you don't have to fork a new thread to use this feature.

Notice that, since there is at most one thread to access the UI of a given desktop, the event listener must NOT be time-consuming. Otherwise, it will block other event listeners from execution. Thus, if you have a long operation to do, you could use event queue's asynchronous event listener, or implement it as a synchronous task and handle lengthy operation outside of the activation block.

Version History

Version	Date	Content
5.0.6	November 2010	This feature was introduced. With 5.0.5 or prior, you have to use Event Queues or Synchronous Tasks.

References

- [1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#schedule\(org.zkoss.zk.ui.Desktop,org.zkoss.zk.ui.EventListener,java.util.List<org.zkoss.zk.ui.event.Event>\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Executions.html#schedule(org.zkoss.zk.ui.Desktop,org.zkoss.zk.ui.EventListener,java.util.List<org.zkoss.zk.ui.event.Event>))
[2] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/EventListener.html#onEvent\(org.zkoss.zk.ui.Event\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/EventListener.html#onEvent(org.zkoss.zk.ui.Event))

Configuration

Since ZK 6, in addition to PollingServerPush [1] and CometServerPush [2], ZK provides a third Server Push implementation - Asynchronous Processing CometServerPush [3]. As their names suggest, they implement the Client-Polling, Comet (aka., long-polling) and Servlet 3 Comet (Servlet 3 Asynchronous Processing-based Comet [4]) server pushes. Client-polling is available in all editions, Comet Server Push is available in ZK PE and EE, while ZK EE supports Servlet 3 Comet Push.

The default implementation depends on which ZK edition you use:

ZK Edition	Technology	ZK Implementation
CE	Client Polling	PollingServerPush [1]
PE	Comet Server Push	CometServerPush [2]
EE	Servlet 3 Comet	CometServerPush [3]

You can also configure ZK to use the one you prefer, or even use a custom server push.

Note that Comet Server Push is available in ZK 5 EE only. By default, ZK 5 CE and ZK 5 PE use PollingServerPush [1], and ZK 5 EE uses CometServerPush [3].

Choose an Implementation

Client-polling is based on a timer that peeks the server continuously to see if any data is to be *pushed* to the client, while Comet establishes a permanent connection for instant *push*. Client-polling will introduce more traffic due to the continuous peeks, but Comet will consume the network connections that a server allows.

Page-level Configuration

You could configure a particular ZK page to use a particular implementation by the use of DesktopCtrl.enableServerPush(org.zkoss.zk.ui.sys.ServerPush) [2]. For example,

```
( (DesktopCtrl)desktop ).enableServerPush( new org.zkoss.zk.ui.impl.PollingServerPush(2000,5000,-1) );
```

Application-level Configuration

If you would like to change the default server push for the whole application, you could use the server-push-class element as follows.

```
<device-config>
    <device-type>ajax</device-type>
    <server-push-class>org.zkoss.zk.ui.impl.PollingServerPush</server-push-class>
</device-config>
```

where you could specify any implementation that implements ServerPush^[5].

Client-Polling Configuration

The client-polling server push is implemented with an implicit timer at the client. The interval of the timer depends on the loading of the server. For example, the interval becomes longer if the time to get a response has become longer. PollingServerPush^[1] uses a timer to peek if the server has any data to *push* back. The period between two peeks is determined by a few factors.

PollingServerPush.delay.min

The minimal delay to send the second polling request (unit: milliseconds).

Default: 1000.

PollingServerPush.delay.max

The maximal delay to send the second polling request (unit: milliseconds).

Default: 15000.

PollingServerPush.delay.factor

- The delay factor. The real delay is the processing time multiplied by the delay factor. For example, if the last request took 1 second to process, then the client polling will be delayed for `1 x factor` seconds. Default: 5.
- The larger the factor, the longer the delay tends to be.

It could be configured in WEB-INF/zk.xml by use of the preference element as follows.

```
<preference>
    <name>PollingServerPush.delay.min</name>
    <value>3000</value>
</preference>
<preference>
    <name>PollingServerPush.delay.max</name>
    <value>10000</value>
</preference>
<preference>
    <name>PollingServerPush.delay.factor</name>
    <value>5</value>
</preference>
<!-- JavaScript code to start the server push; rarely required
<preference>
```

```
<name>PollingServerPush.start</name>
<value></value>
</preference>
<preference>
    <name>PollingServerPush.stop</name>
    <value></value>
</preference>
-->
```

In additions, you could specify them in the constructor: int, int) PollingServerPush.PollingServerPush(int, int, int)^[6]. For example,

```
((DesktopCtrl) desktop).enableServerPush(
    new org.zkoss.zk.ui.impl.PollingServerPush(2000, 10000, 3));
```

Comet Server Push Config

The comet server push is implemented with a pre-established and 'virtual' permanent connection. It is like sending a taxi to the server, and waiting in the server until there is data to send back. Meanwhile, the client-polling server push is like sending a taxi periodically to the server, and leaving immediately if no data is available.

CometServerPush^[2] comes with its own configuration parameters such as retry delay, retry count, and ajax timeout. Configure these parameters by using the preference element in your WEB-INF/zk.xml e.g.

```
<preference>
    <name>CometServerPush.retry.delay</name>
    <value>3000</value><!-- 3 seconds delay between each retry-->
</preference>
<preference>
    <name>CometServerPush.retry.count</name>
    <value>3</value><!-- 3 tries for each request -->
</preference>
<preference>
    <name>CometServerPush.ajax.timeout</name>
    <value>180000</value><!-- 3 minutes -->
</preference>
```

CometServerPush.retry.delay

The minimum time delay to send the next comet request retry (unit: milliseconds).

Default: 5000.

CometServerPush.retry.count

The maximum retry counts if a comet request fails, -1 means keep retry forever (unit: count).

Default: 10.

CometServerPush.ajax.timeout

The amount of time a comet request will wait for a server response until it aborts. (unit: milliseconds).

Default: varies, depending on browsers [7]

Error Handling

The configuration of the errors is handled by the client-reload element, specified in WEB-INF/zk.xml. The markup below demonstrates an example of catching an error of the server push:

```
<error-reload>
  <device-type>ajax</device-type>
  <connection-type>server-push</connection-type>
  <error-code>410</error-code>
  <reload-uri>/login.zul</reload-uri>
</error-reload>
```

where the connection-type element specifies through which channel the requests are sent. By default it is the AU channel in which Ajax requests are sent by widgets running at the client. If you would like to specify the error page for server-push then connection-type must be set to server-push.

"dummy" Requests

When using ServerPush the client engine will send "dummy"-requests (without extra payload) to the /zkau servlet to pull queued server side updates.

Those requests look similar to this:

```
dtid=z_m06&cmd_0=dummy&opt_0=i
```

In the case of PollingServerPush there will be one of these requests per configured interval, sometimes causing a firewall to give a false alert.

In the case of CometServerPush, after a long polling comet request is returned, there will be one "dummy" request sent to retrieve server update.

WebSocketServerPush

Since ZK 8.5.0, the WebSocket based Server Push WebSocketServerPush [8] is provided. When WebSocket connection is enabled, this will be used as default, other configuration specifying which Server Push to use will be ignored.

Version History

Version	Date	Content
6.0.0	Feb 2012	The CometServerPush is available in ZK PE, while ZK EE supports Servlet 3 Asynchronous Processing-based Comet.

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/impl/PollingServerPush.html#>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkex/ui/comet/CometServerPush.html#>
- [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkmax/ui/comet/CometServerPush.html#>
- [4] http://books.zkoss.org/wiki/Small_Talks/2012/February/
New_Features_of_ZK_6#ZK_Comet_supports_Servlet_3_Asynchronous_Processing
- [5] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/ServerPush.html#>
- [6] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/impl/PollingServerPush.html#PollingServerPush\(int,](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/impl/PollingServerPush.html#PollingServerPush(int,)
- [7] <https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/timeout>
- [8] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkmax/au/websocket/WebSocketServerPush.html#>

Clustering

ZK components, pages, and desktops are all serializable, so using ZK in a clustering environment is straightforward. However, it is a challenge to develop a sophisticated application that is ready for clustering.

Here we discuss how to configure ZK for a clustering environment, how to configure it for some servers, and the important notes when developing a clustering-ready application with ZK.

ZK Configuration

Turn on Serializable UI Factory

Notice: Required

To use ZK in a clustering environment, you have to use the serializable UI factory. It could be done by specifying the following statement in WEB-INF/zk.xml:

```
<system-config>
    <ui-factory-class>org.zkoss.zk.ui.http.SerializableUiFactory</ui-factory-class>
</system-config>
```

SerializableUiFactory^[1] is the UI factory that will instantiate serializable sessions such that the sessions, components, pages, and desktops will be serialized when a session is about to deactivate.

Additional settings for various servers

Please refer to the following links for detailed settings.

- Tomcat Cluster
- WebLogic Cluster
- Google App Engine
- JBoss Cluster

Turn on Log

[Optional]

If an attribute or a listener is not serializable, ZK will skip it, i.e., not serialize it (similar to how a Servlet container serializes the attributes of sessions). It is sometimes hard to know what is ignored since it is common for a developer to forget to declare a value or a listener as serializable.

To detect this problem, you can turn on the logger for org.zkoss.io.serializable to the DEBUG level (). Please read Logger for details.

Disable the Use of zscript

[Optional]

The interpreter (BeanShell) does not work well under the clustering environment. Since the serialization is not stable, zscript cannot be used in a clustering environment. To avoid accidental or unintended use it is recommended to disable zscript in your ZK application with the following configuration (in zk.xml):

```
<system-config>
    <disable-zscript>true</disable-zscript>
</system-config>
```

Configuration Not Allowed

Here is a list of configurations that can not be used in the clustering environment. They are disabled by default. However, it is worth to double check that none are enabled accidentally.

Event Processing Thread

Do not enable the event processing thread. The event processing thread might be suspended, while the (suspended) thread cannot be migrated from one machine to another.

It is disabled by default. For more information, please refer to the Event Threads section.

Global Desktop Cache

Do not use GlobalDesktopCacheProvider^[2] (global desktop cache). The global desktop cache is stored in the servlet context, while only the data stored in sessions is migrated when failover takes place.

The default is SessionDesktopCacheProvider^[3] instead of desktop-scoped desktop cache. Just make sure you don't configure it wrong.

Version History

Version	Date	Content
5.0.7	April 2011	The log called <code>org.zkoss.io.serializable</code> was introduced.
5.0.8	June 2011	The listener called <code>org.zkoss.zkplus.cluster.ClusterSessionPatch</code> was introduced.

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/http/SerializableUiFactory.html#>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/impl/GlobalDesktopCacheProvider.html#>
- [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/impl/SessionDesktopCacheProvider.html#>

Server Configuration

The configuration of a Web server really depends on the server itself. There is no standard approach.

Load Balancer (Required)

Be sure to configure sticky session^[1] on your load balancer because ZK only works correctly on this setting.

Check ZK DesktopImpl.java, you will see there are lots of `synchronized` used in many methods including:

- generate desktop ID
- addPage(), removePage()
- enableServerPush()
- processing server push update

The keyword `synchronized` only works in a single JVM, so it's impossible to keep the same state between 2 different nodes with session replication if 2 nodes receive requests at roughly the same time.

Each AU request changes a Desktop's status of a session, if 2 AU requests are processed by 2 different clustering nodes and each one has its own session, then one session state might override another session.

Apache + Tomcat

For configuring Apache + Tomcat, please refer to

- How to Run ZK on Apache + Tomcat clustering, Part I
- How to Run ZK on Apache + Tomcat clustering, Part II

More detail settings

- Tomcat Cluster

Google App Engine

For configuring Google App Engine, please refer to

- ZK Installation Guide: Google App Engine

Apache + JBoss

For configuring JBoss, please refer to

- JBoss Cluster

Glassfish

For configuring Glassfish, please refer to

- Glassfish Cluster

WebLogic

For configuring WebLogic, please refer to

- WebLogic Cluster

References

[1] <http://wiki.metawerx.net/wiki/StickySessions>

Programming Tips

Objects Referenced by UI Must be Serializable

Objects that are referenced by a UI object, such as components and pages, have to be serializable. Otherwise, they might have no value after de-serialized, or cause an exception (depending on how it is used).

Attributes of UI Objects

If the value of an attribute is not serializable, it will be ignored. Thus, it will become null after de-serialized. So, it is better to make them all serializable (such as implementing `java.io.Serializable`), or handle the serialization manually (refer to the Clustering Listeners section below) .

zscript

It is OK, though not recommended, to use zscript in a clustering environment, but there are some limitations.

- BeanShell's function is not serializable. For example, the following won't work:

```
void foo() {  
}
```

- The value of variables must be serializable

Notice that it is not recommended to use zscript in the clustering environment. After all, the performance of BeanShell is not good.

Event Listeners

Event listeners have to be serializable. Otherwise, it will be ignored after serialization.

The simplest way to make an event listener serializable is to implement `SerializableEventListener`^[2] (available since 5.0.6), instead of `EventListener`^[1].

For example,

```
button.addEventListerner(Events.ON_CLICK,  
    new SerializableEventListerner() {  
        public void onEvent(Event event) {  
            ....  
        }  
    } );
```

Data Models

The data models, such as `ListModel`^[7] and `ChartModel`^[1], have to be serializable. Otherwise, the UI object (such as grid) won't behave correctly. The implementations provided by ZK are serializable. However, the items to be stored in the data models have to be serializable too.

Composers

If you extend from ZK built-in composer like `GenericAutowireComposer`^[1], or `SelectorComposer`^[3] you have to make sure all of its members are serializable (or transient), since the implementation will keep a reference in the applied component.

When implementing from `Composer`^[2] directly, the composer could be non-serializable if you don't keep a reference in any UI object. In other words, the composer will be dropped after `Composer.doAfterCompose(org.zkoss.zk.ui.Component)`^[3]

ViewModels

If you are using ZK MVVM then your `ViewModel` classes must be serializable.

Clustering Listeners

If there are non-serializable objects, you could implement one of the clustering listeners to handle them manually as described below. Basically, there are two kinds of clustering listeners for different purposes:

- **Serialization Listeners:** they are called when an object is about to be serialized, and after it has been de-serialized.
Example: `ComponentSerializationListener`^[4] and `PageSerializationListener`^[5]
- **Activation Listeners:** they are called when a session is about to be passivated, and after it has been activated.
Examples: `ComponentActivationListener`^[6] and `PageActivationListener`^[7].

To register a listener is straightforward: just implement the corresponding listener interface. For example, you could implement `ComponentActivationListener`^[6] if an object is stored in a component and wants to be called on activation and passivation.

Passivation Flow

When a session is about to be passivated (such as moving to another machine), the activation listeners will be called first to notify the passivation, and then the serialization listeners will be called before the object is serialized.

Sequence	Description
1	Invokes <code>SessionActivationListener.willPassivate(org.zkoss.zk.ui.Session)</code> ^[8] for each object referenced by the Session ^[2] that will be passivated
2	Invokes <code>DesktopActivationListener.willPassivate(org.zkoss.zk.ui.Desktop)</code> ^[9] for each object referenced by each Desktop ^[9] that will be passivated
3	Invokes <code>PageActivationListener.willPassivate(org.zkoss.zk.ui.Page)</code> ^[10] for each object referenced by each Page ^[8] that will be passivated
4	Invokes <code>ComponentActivationListener.willPassivate(org.zkoss.zk.ui.Component)</code> ^[11] for each object referenced by each Component ^[1] that will be passivated
5	Invokes <code>SessionSerializationListener.willSerialize(org.zkoss.zk.ui.Session)</code> ^[12] for each object referenced by the Session ^[2] that will be passivated

6	Serializes the session
7	Invokes DesktopSerializationListener.willSerialize(org.zkoss.zk.ui.Desktop) ^[13] for each object referenced by each Desktop ^[9] that will be passivated
8	Serializes desktops of the session
9	Invokes PageSerializationListener.willSerialize(org.zkoss.zk.ui.Page) ^[14] for each object referenced by each Page ^[8] that will be passivated
10	Serializes pages of each desktop
11	Invokes ComponentSerializationListener.willSerialize(org.zkoss.zk.ui.Component) ^[15] for each object referenced by each Component ^[1] that will be passivated
12	Serializes components of each page

Activation Flow

When a session is about to be activated (such as moving from another machine), the serialization listener is called after the object has been deserialized. After all objects are deserialized, the activation listener will be called to notify a session has been activated.

Sequence	Description
1	Deserializes the session
2	Deserializes desktops of the session
3	Deserializes pages of each desktop
4	Deserializes components of each page
5	Invokes ComponentSerializationListener.didDeserialize(org.zkoss.zk.ui.Component) ^[16] for each object referenced by each Component ^[1] that will be passivated
6	Invokes PageSerializationListener.didDeserialize(org.zkoss.zk.ui.Page) ^[17] for each object referenced by each Page ^[8] that will be passivated
7	Invokes DesktopSerializationListener.didDeserialize(org.zkoss.zk.ui.Desktop) ^[18] for each object referenced by each Desktop ^[9] that will be passivated
8	Invokes SessionSerializationListener.didDeserialize(org.zkoss.zk.ui.Session) ^[19] for each object referenced by the Session ^[2] that will be passivated
9	Invokes SessionActivationListener.didActivate(org.zkoss.zk.ui.Session) ^[20] for each object referenced by the Session ^[2] that will be passivated
10	Invokes DesktopActivationListener.didActivate(org.zkoss.zk.ui.Desktop) ^[21] for each object referenced by each Desktop ^[9] that will be passivated
11	Invokes PageActivationListener.didActivate(org.zkoss.zk.ui.Page) ^[22] for each object referenced by each Page ^[8] that will be passivated
12	Invokes ComponentActivationListener.didActivate(org.zkoss.zk.ui.Component) ^[23] for each object referenced by each Component ^[1] that will be passivated

Working Thread Cannot Last Two or More Requests

Since the thread cannot be migrated from one machine to another, you couldn't use a working thread that works across multiple requests. For example, you cannot start a working thread in one request, and then invoke it in another request, since the session might be passivated between the requests.

It also implies you cannot use a working thread to handle a long operation. Rather, you have to use the so-called Echo Event.

Users of ZK 5.0.5 or prior cannot deploy the event queues for the session and application scope. However, users of ZK 5.0.6 or later have no such limitation.

Debugging Tips

`System.setProperty("sun.io.serialization.extendedDebugInfo", "true")` can print detailed debugging information about which member field is not serializable when a `java.io.NotSerializableException` happens.

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/GenericAutowireComposer.html#>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Composer.html#>
- [3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Composer.html#doAfterCompose\(org.zkoss.zk.ui.Component\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Composer.html#doAfterCompose(org.zkoss.zk.ui.Component))
- [4] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ComponentSerializationListener.html#>
- [5] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/PageSerializationListener.html#>
- [6] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ComponentActivationListener.html#>
- [7] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/PageActivationListener.html#>
- [8] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/SessionActivationListener.html#willPassivate\(org.zkoss.zk.ui.Session\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/SessionActivationListener.html#willPassivate(org.zkoss.zk.ui.Session))
- [9] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/DesktopActivationListener.html#willPassivate\(org.zkoss.zk.ui.Desktop\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/DesktopActivationListener.html#willPassivate(org.zkoss.zk.ui.Desktop))
- [10] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/PageActivationListener.html#willPassivate\(org.zkoss.zk.ui.Page\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/PageActivationListener.html#willPassivate(org.zkoss.zk.ui.Page))
- [11] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ComponentActivationListener.html#willPassivate\(org.zkoss.zk.ui.Component\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ComponentActivationListener.html#willPassivate(org.zkoss.zk.ui.Component))
- [12] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/SessionSerializationListener.html#willSerialize\(org.zkoss.zk.ui.Session\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/SessionSerializationListener.html#willSerialize(org.zkoss.zk.ui.Session))
- [13] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/DesktopSerializationListener.html#willSerialize\(org.zkoss.zk.ui.Desktop\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/DesktopSerializationListener.html#willSerialize(org.zkoss.zk.ui.Desktop))
- [14] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/PageSerializationListener.html#willSerialize\(org.zkoss.zk.ui.Page\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/PageSerializationListener.html#willSerialize(org.zkoss.zk.ui.Page))
- [15] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ComponentSerializationListener.html#willSerialize\(org.zkoss.zk.ui.Component\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ComponentSerializationListener.html#willSerialize(org.zkoss.zk.ui.Component))
- [16] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ComponentSerializationListener.html#didDeserialize\(org.zkoss.zk.ui.Component\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ComponentSerializationListener.html#didDeserialize(org.zkoss.zk.ui.Component))
- [17] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/PageSerializationListener.html#didDeserialize\(org.zkoss.zk.ui.Page\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/PageSerializationListener.html#didDeserialize(org.zkoss.zk.ui.Page))
- [18] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/DesktopSerializationListener.html#didDeserialize\(org.zkoss.zk.ui.Desktop\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/DesktopSerializationListener.html#didDeserialize(org.zkoss.zk.ui.Desktop))
- [19] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/SessionSerializationListener.html#didDeserialize\(org.zkoss.zk.ui.Session\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/SessionSerializationListener.html#didDeserialize(org.zkoss.zk.ui.Session))
- [20] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/SessionActivationListener.html#didActivate\(org.zkoss.zk.ui.Session\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/SessionActivationListener.html#didActivate(org.zkoss.zk.ui.Session))
- [21] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/DesktopActivationListener.html#didActivate\(org.zkoss.zk.ui.Desktop\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/DesktopActivationListener.html#didActivate(org.zkoss.zk.ui.Desktop))
- [22] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/PageActivationListener.html#didActivate\(org.zkoss.zk.ui.Page\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/PageActivationListener.html#didActivate(org.zkoss.zk.ui.Page))
- [23] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ComponentActivationListener.html#didActivate\(org.zkoss.zk.ui.Component\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ComponentActivationListener.html#didActivate(org.zkoss.zk.ui.Component))

Integration

This chapter describes how to integrate ZK with other frameworks, including how to write a JSP/JSF tag with ZK components, how to access ZK components in foreign Ajax channel, how to work with form-based framework, and so on.

Accessing Java EE Scope Objects

Executions

org.zkoss.zk.ui.Executions^[1]

getCurrent

Executions.getCurrent()^[2]

Retrieves the current execution which contains HTTP request/response.

get HttpServletRequest

```
HttpServletRequest req =  
(HttpServletRequest) Executions.getCurrent().getNativeRequest()
```

getNativeRequest()^[1]

Sessions

org.zkoss.zk.ui.Sessions^[5]

Get Current Session

Sessions.getCurrent()^[6]

Retrieves the current ZK-wrapped session.

Get HttpSession

```
HttpSession nativeSession = (HttpSession)  
Sessions.getCurrent().getNativeSession();
```

References

[1] [https://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#getNativeRequest\(\)](https://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Execution.html#getNativeRequest())

Presentation Layer

In following section, we will describe how to integrate the below frameworks in presentation layer which are responsible for navigating users between pages and presenting data to users, like JSP and Struts.

Bootstrap

Bootstrap ^[1] is a very popular front-end framework for building responsive, mobile-first sites. It can be a good companion for ZK with the following usages:

- Layout a page with the responsive grid system ^[2]
- Style a page with utility CSS classes ^[3]

See the integration example: admin template ^[4]

Include JAR

To avoid downloading bootstrap manually, it's convenient to include it by WebJars ^[5] with Maven.

```
<dependency>
    <groupId>org.webjars</groupId>
    <artifactId>bootstrap</artifactId>
    <version>4.6.0</version>
</dependency>
```

Include CSS

Page Scope

```
<link rel="stylesheet" href="/webjars/bootstrap/4.6.0/css/bootstrap.min.css"?>
```

Application Scope

```
<?xml version="1.0" encoding="UTF-8"?>
<language-addon>
    <addon-name>bootstrap</addon-name>
    <language-name>xul/html</language-name>
    <stylesheet href="/webjars/bootstrap/4.6.0/css/bootstrap.min.css" type="text/css"/>
</language-addon>
```

See ZK Client-side Reference/Language Definition/stylesheet

References

- [1] <https://getbootstrap.com/docs/4.6/getting-started/introduction/>
- [2] <https://getbootstrap.com/docs/4.6/layout/grid/>
- [3] <https://getbootstrap.com/docs/4.6/utilities/borders/>
- [4] <https://github.com/zkoss-demo/admin-template>
- [5] <https://www.webjars.org/>

JSP

Employment/Purpose

Basically there are two approaches to use ZK in JSP pages.

1. Use <jsp:include> to include a ZUL page.
2. Use ZK JSP Tags^[4] in a JSP page directly.

Here we discuss the general concepts applicable to both approaches. For information of ZK JSP Tags, please refer to ZK JSP Tags Essentials. It is also worth to take a look at the HTML Tags section.

Prerequisite

DOCTYPE

To use ZK components correctly, the JSP page must specify DOCTYPE as follows.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html>
 ...

```

BODY Style

By default, ZK will set the CSS style of the BODY tag to width:100%;height:100%. If you prefer to have the browser to decide the height (i.e., the browser's default) for you, you could specify height:auto to the BODY tag (optional).

```
<body style="height:auto">
 ...

```

Browser Cache

Though optional, it is suggested to disable the browser to cache the result page. It can be done as follows.

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Pragma" content="no-cache" />
    <meta http-equiv="Expires" content="-1" />
```

In addition, you could invoke the following statement in JSP to tell ZK to drop desktops once the user navigates to other URL. It is optional but it saves memory since the browser page is not cached and safe to remove if the user

navigates away.

```
<%
    request.setAttribute(org.zkoss.zk.ui.sys.Attributes.NO_CACHE,
Boolean.TRUE);
%>
```

Notice that it has to be invoked before ZK JSP's zkhead tag, if ZK JSP is used, or before the first jsp:include that includes a ZUL page.

HTML Form

ZK input components (datebox, slider, listbox and so on) work seamlessly with HTML form. In addition to Ajax, you could process input in batch with Servlets.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<%@ taglib uri="http://www.zkoss.org/jsp/zul" prefix="z" %>

<html>
  <body>
    <z:page>
      <form action="/foo/legacy">
        <table>
          <tr>
            <td>When</td><td><z:datebox name="when"/></td>
          </tr>
          <tr>
            <td>Which></td>
            <td>
              <z:listbox name="which">
                <z:listitem label="choice 1"/>
                <z:listitem label="choice 2"/>
              </z:listbox>
            </td>
          </tr>
          <tr>
            <td><z:button type="submit" label="Submit"/></td>
            <td><z:button type="reset" label="Reset"/></td>
          </tr>
        </form>
      </z:page>
    </body>
  </html>
```

The name Property

If you want to submit the values of the ZK components, you have to place the component inside the form and then specify the `name` property. Thus, when the form is submitted, the value of, say, the datebox will be sent together with the name you specified. For example,

Submit			
When	Nov 10, 2010 <input type="button" value=""/>	Name	Mark Gates
Department	Manufactory <input type="button" value=""/>	Type	New Average
<input type="button" value="Submit"/>			

```
<window title="Submit" border="normal" xmlns:n="native">
  <n:form action="/fooLegacy">
    <grid>
      <rows>
        <row>
          When
          <datebox name="when" />
        </row>
        <row>
          Name
          <textbox name="name" />
        </row>
        <row>
          Department
          <combobox name="department">
            <comboitem label="RD" />
            <comboitem label="Manufactory" />
            <comboitem label="Logistics" />
          </combobox>
        </row>
        <row>
          Type
          <listbox name="type">
            <listitem label="New" value="new"/>
            <listitem label="Average" value="avarage"/>
          </listbox>
        </row>
        <row>
          <button type="submit" label="Submit"/>
        </row>
      </rows>
    </grid>
  </n:form>
</window>
```

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

```
?when=Nov+10%2C+2010&name=Mark+Gates&department=Manufactory&type=new
```

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

Components that Support the name Property

All input-types components support the name property, such as textbox, datebox, decimalbox, intbox, combobox, bandbox, slider and calendar.

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

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

name	gender
Mary	FEMALE
John	MALE
Jane	FEMALE
Henry	MALE

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

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

Notice that, to use the list boxes and tree controls with the name property, you have to specify the value property for listitem and treeitem, respectively. They are the values being posted to the servlets.

Rich User Interfaces

Because a `form` component could contain any kind of components, the rich user interfaces could be implemented independently of the existent servlets. For example, you could listen to the `onOpen` event and fulfill a tab panel as illustrated in the previous sections. Yet another example, you could dynamically add more rows to a grid control, where each row might control input boxes with the `name` property. Once user submits the form, the most updated content will be posted to the servlet.

Communication with zul

Pass Data to zul

Query String

If the data you pass can be visible to users, you can just pass it through a query string.

Just put a hyperlink on a page: ``

or

```
call      HttpServletResponse.sendRedirect(req.getContextPath() +  
"/redirected.zul?myparam=myvalue") in a Servlet.
```

In ZK, you get the passed data with `Executions.getCurrent().getParameter("mykey")`

Session Attributes

```
HttpServletRequest.getSession().setAttribute("myKey", "myValue")
```

Pass Data to JSP

Query String

Just put a hyperlink on a page: ``

or

```
call Executions.getCurrent().sendRedirect("/redirected.zul?myparam=myvalue") in a  
ZK Controller.
```

Session Attributes

```
Sessions.getCurrent().setAttribute("myKey", "myValue")
```

Struts

The use of Struts [1] with ZK is straightforward: just replace JSP pages with ZUL pages. You don't need to modify action handlers, data models and others. All you need to do is to map the result view to a ZUL page instead of JSP. In addition, EL expressions will work the same way even in the ZUL page.

Use ZUL instead of JSP

First, let us take the Hello World example in Struts tutorial [2] as an example. We could provide a ZUL page called `HelloWorld.zul` to replace `HelloWorld.jsp` as follows.

```
<?page title="Hello World!"?>

<h: h2 xmlns:h="xhtml">
${messageStore.message}
</h: h2>
```

As shown, you could use the same EL expression to access the data provided by Struts and your action handler.

Then, you map the `hello` action to `HelloWorld.zul` by modifying `WEB-INF/classes/struts.xml` as follows.

```
<action name="hello" class="org.apache.struts.helloworld.action.HelloWorldAction" method="execute">
    <result name="success">/HelloWorld.zul</result>
</action>
```

Then, you could visit `http://localhost:8080/Hello_World_Struts2_Ant/hello.action` as you are used to and have the same result.

Of course, it is a ZUL document. You could have any Ajax behavior you'd like.

Access Data Model of Struts in Composer

The data (so-called model) provided by Struts (or the action) can be retrieved by invoking `Execution.getAttribute(java.lang.String)` [4]. For example,

```
package foo;
import org.zkoss.zk.ui.util.Composer;
import org.zkoss.zk.ui.*;
import org.zkoss.zul.*;
import org.apache.struts.helloworld.model.MessageStore;

public class FooComposer implements org.zkoss.zk.ui.util.Composer {
    public void doAfterCompose(Component comp) {
        MessageStore mstore =
        Executions.getCurrent().getAttribute("messageStore");
        comp.appendChild(new Label(": "+mstore.getMessage()));
    }
}
```

Submit Form

By replacing JSP with ZUML, you could enable a *static* page with ZK's power. And, you could do what any ZUML documents can do. In other words, Struts is used only for Model and Controller, while ZK for View. However, sometimes you have to redirect back to submit-based URL (maybe another action with parameters). It can be done easily by enclosing the input components with HTML FORM. For example,

```
<?taglib uri="http://www.zkoss.org/dsp/web/core" prefix="c"?>
<n:form action="${c:encodeURL('/login.action')}" method="POST" xmlns:n="native">
<grid>
  <rows>
    <row>
      User: <textbox name="user"/>
    </row>
    <row>
      Password: <textbox name="password"/>
    </row>
    <row>
      <button label="Login" type="submit"/>
    </row>
  </rows>
</grid>
</n:form>
```

As shown above, notice that

- Every input (including listbox and tree) shall be assigned with a name that will become the parameter's name when submitting the form.
- You could use the encodeURL function to encode an URL.

For more information, please refer to ZK Developer's Reference/Integration/Use_ZK_in_JSP#HTML_Form the Use ZK in JSP section.

Avoid Filtering ZK AU Requests

When adopting Struts, we usually apply its filter to all URL in web.xml like:

```
<filter>
  <filter-name>struts2</filter-name>
  <filter-class>
    org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
  </filter-class>
</filter>
<filter-mapping>
  <filter-name>struts2</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

But this filter will also intercept ZK AU requests and make ZK components works abnormally. You might see a similar error message like:

```
There is no Action mapped for namespace / and action name zkau. - [unknown location]
```

To avoid this problem, you can add the line below in `struts.xml` to exclude ZK au requests for struts filter

```
<struts>
  <constant name="struts.action.excludePattern" value="/zkau"/>
  <!-- other configurations -->
</struts>
```

References

- [1] <http://struts.apache.org/>
- [2] <http://struts.apache.org/2.x/hello-world-using-struts-2.html>

Portal

Configuration

Here we describe the standard configuration. Depending on the portal server, you might have more than one configuration to set. For more information, please refer to ZK Installation Guide.

WEB-INF/portlet.xml

To use it, you first have to add the following portlet definition for DHmlLayoutPortlet^[1] into WEB-INF/portlet.xml. Notice that expiration-cache must be set to zero to prevent portals from caching the result.

```
<portlet>
  <description>ZK loader for ZUML pages</description>
  <portlet-name>zkPortletLoader</portlet-name>
  <display-name>ZK Portlet Loader</display-name>

  <portlet-class>org.zkoss.zk.ui.http.DHmlLayoutPortlet</portlet-class>

  <expiration-cache>0</expiration-cache>

  <supports>
    <mime-type>text/html</mime-type>
    <portlet-mode>VIEW</portlet-mode>
  </supports>

  <supported-locale>en</supported-locale>

  <portlet-info>
    <title>ZK</title>
    <short-title>ZK</short-title>
    <keywords>ZK, ZUML</keywords>
  </portlet-info>
```

```
</portlet-info>
</portlet>
```

WEB-INF/web.xml

ZK portlet loader actually delegates the loading of ZUML documents to ZK Loader (DHtmlLayoutServlet^[2]). Thus, you have to configure `WEB-INF/web.xml` as specified in ZK Installation Guide, even if you want to use only portlets.

Use ZK Portlet

The `zk_page` and `zk_richlet` Parameter and Attribute

ZK portlet loader is a generic loader. To load a particular ZUML page, you have to specify either a request parameter, a portlet attribute or a portlet preference called `zk_page`, if you want to load a ZUML page, or `zk_richlet`, if you want to load a richlet.

More precisely, ZK portlet loader first checks the following locations for the path of the ZUML page or the richlet. The lower the number, the higher the priority.

1. The request parameter (`RenderRequest's getParameter`) called `zk_page`. If found, it is the path of the ZUML page.
2. The request attribute (`RenderRequest's getAttribute`) called `zk_page`. If found, it is the path of the ZUML page.
3. The request preference (`RenderRequest's getPortletPreferences's getValue`) called `zk_page`. If found, it is the path of the ZUML page.
4. The request parameter (`RenderRequest's getParameter`) called `zk_richlet`. If found, it is the path of the richlet.
5. The request attribute (`RenderRequest's getAttribute`) called `zk_richlet`. If found, it is the path of the richlet.
6. The request preference (`RenderRequest's getPortletPreferences's getValue`) called `zk_richlet`. If found, it is the path of the richlet.
7. The initial parameter (`PortletConfig's getInitParameter`) called `zk_page`. If found, it is the path of the ZUML page.

Examples

How to pass a request parameter or attribute to a portlet depends on the portal. You have to consult the user's guide of your favorite portal for details, or refer to ZK Installation Guide.

References

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

ZK Filter

If you prefer to Ajax-ize a dynamically generated HTML page (e.g., the output of a Velocity Servlet), you could use ZK Filter to process the generated page. The content of the generated page will be interpreted by ZK Filter as a ZUML document. Thus, please make sure the output is a valid ZUML document, such as it must be a valid XML. If the output is HTML, it must be a valid XHTML document.

To enable ZK Filter, you have to configure WEB-INF/web.xml, as shown below.

```
<filter>
    <filter-name>zkFilter</filter-name>
    <filter-class>org.zkoss.zk.ui.http.DHtmlLayoutFilter</filter-class>
    <init-param>
        <param-name>extension</param-name>
        <param-value>html</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>zkFilter</filter-name>
    <url-pattern>/my/dyna.jsp</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>zkFilter</filter-name>
    <url-pattern>/my/dyna/*</url-pattern>
</filter-mapping>
```

where url-pattern is the servlets that you would like ZK Filter to process.

The extension parameter (init-param) defines the language of the dynamical output. By default, it is html, since most legacy servlets generate an HTML document. If the output is a ZUL document, you could specify zul as the extension.

Notice that, if you want to filter the output from include and/or forward, remember to specify the dispatcher element with REQUEST and/or INCLUDE. Consult the Java Servlet Specification for details. For example,

```
<filter-mapping>
    <filter-name>zkFilter</filter-name>
    <url-pattern>/my/dyna/*</url-pattern>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>INCLUDE</dispatcher>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>ERROR</dispatcher>
</filter-mapping>
```

Performance Consideration for Filtering XHTML

If the extension is html (and the output is XHTML), it means each HTML tag will be converted to an XHTML component. It is convenient if you want to manipulate them dynamically. However, it costs more memory since ZK has to maintain the states of each HTML tag. Thus, it is suggested to use the native namespace for the portion whose content won't be changed dynamically.

ZK Filter versus UI Factory

ZK Filter is designed to handle the output of a legacy servlet. If you would like to load a ZUML document from resources other than Web pages, such as from the database, you could implement UiFactory^[1]. It is generally done by extending from AbstractUiFactory^[2] and overriding java.lang.String) AbstractUiFactory.getPageDefinition(org.zkoss.zk.ui.sys.RequestInfo, java.lang.String)^[3]. For more information, please refer to ZK Configuration Reference.

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/UiFactory.html#>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/impl/AbstractUiFactory.html#>
- [3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/impl/AbstractUiFactory.html#getPageDefinition\(org.zkoss.zk.ui.sys.RequestInfo, java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/impl/AbstractUiFactory.html#getPageDefinition(org.zkoss.zk.ui.sys.RequestInfo, java.lang.String))

Foreign Templating Framework

Employment/Purpose

Here we describe how to make a ZUL page to be assembled at the client by using Ajax to request ZUL pages separately in a foreign templating framework^[1].

You could skip this chapter if you'd like to use ZK's templating technology, such as Templating: composition, Servlet's inclusion (javax.servlet.RequestDispatcher's include) and macro components.

ZK also supports many powerful layout components, such as portallayout, borderlayout, tablelayout, columnlayout and so on^[2]. You could use them to have similar or better effect, and skip this chapter.

[1] Apache Tiles (<http://tiles.apache.org/>) is a typical templating framework and allows developers to assemble UI at both server and client.

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

Prerequisite

DOCTYPE

To use ZK components correctly, the templating page must specify DOCTYPE as follows.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html>
  ...

```

Browser Cache

Though optional, it is suggested to disable the browser to cache the result page. It can be done as follows.

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Pragma" content="no-cache" />
    <meta http-equiv="Expires" content="-1" />
```

Make a ZUL page as a fragment

Include a ZUL page when receiving a request

By default, if a ZUL page is requested by the browser directly, it will generate a complete HTML structure, including HTML, HEAD and BODY tags. On the other hand, if the assembling is done by inclusion (`javax.servlet.RequestDispatcher's include`), a ZUL page will be generated as a HTML fragment without HTML, HEAD, and BODY. For example, if a ZUL page is included by `jsp:include`, then it won't generate HTML/HEAD/BODY, such that the following JSP page will be rendered correctly.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<%-- a JSP page --%>
<html>
  <body>
    <jsp:include page="frag.zul"/>
  ...

```

In other words, if the result page is assembled when the request is received, you don't need to do anything specially^[1]. However, if the assembling is done at the client side by using Ajax to request fragments after loaded, you have to read the following section.

[1] You might take a look at Use ZK in JSP for more information.

Load a ZUL page with an Ajax request

As described above, if a ZUL page is requested by the browser directly, it will, by default, generate a complete HTML structure, including HTML, HEAD and BODY tags. To disable it, you could specify a special parameter called `zk.redrawCtrl=page`. For example, you might have a HTML page that loads a ZUL page at the client with jQuery as follows.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
    <title>Mash-up of ZUML apges</title>
    <script src="http://code.jquery.com/jquery-1.4.2.min.js">
    </script>
  </head>
  <body>
    <div id="anchor"></div>
    <button onclick="$('#anchor').load('foo.zul?zk.redrawCtrl=page')">Load the fragment</button>
  </body>

```

```
</html>
```

The `zk.redrawCtrl` parameter is used to control how a ZUL page is specified. In this case, since `page` is specified, the generation of HTML, HEAD and BODY tags are disabled.

Alternative: using the request-scoped attribute called `org.zkoss.zk.ui.page.redrawCtrl`

If a ZUL page is always loaded as a fragment by the client, you could specify the request-scoped attribute called `org.zkoss.zk.ui.page.redrawCtrl` (`Attributes.PAGE_REDRAW_CONTROL` (http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/Attributes.html#PAGE_REDRAW_CONTROL)) with `page`, such that the generation of HTML, HEAD and BODY tags are always disabled no matter if the `zk.redrawCtrl` parameter is specified or not.

For example,

```
<window title="whatever content you want"/>
<custom-attributes scope="request" org.zkoss.zk.ui.page.redrawCtrl="page"/>
...
</window>
```

Then, you don't need to specify the `zk.redrawCtrl` parameter when loading it at the client (e.g., `$('#anchor').load('foo.zul')`).

Of course, if the fragment itself is a JSP page and then use inclusion to include a ZUL page (or use ZK JSP Tags), then the generated HTML structure is already a correct HTML fragment (and you don't need to anything described above).

Server-side memory optimization: turn off browser cache

As described in [ZK in JSP (http://books.zkoss.org/wiki/ZK_Developer's_Reference/Integration/Use_ZK_in_JSP#Browser_CacheUsage)], the memory footprint at the server can be improved by turning off the browser cache for the HTML page that will load ZUL pages later. For example, we could add `no-cache` and `expires` as follows (line 4 and 5):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
    <meta http-equiv="Pragma" content="no-cache" />
    <meta http-equiv="Expires" content="-1" />
    <title>Mash-up of ZUML apges</title>
    <script src="http://code.jquery.com/jquery-1.4.2.min.js">
    </script>
  </head>
  <body>
    <div id="#anchor"></div>
    <button onclick="$('#anchor').load('foo.zul')>Load the fragment</button>
  </body>
</html>
```

In addition, we have to specify a request-scoped attribute called `org.zkoss.zk.desktop.nocache` in the ZUL page being loaded as follows:

```
<window title="whatever content you want"/>
<custom-attributes scope="request" org.zkoss.zk.desktop.nocache="true"
org.zkoss.zk.ui.page.redrawCtrl="page"/>
...
</window>
```

Note: since 5.0.8, assigning page to the zk.redrawCtrl parameter implies *no-cache*, i.e., zk.redrawCtrl=page implies org.zkoss.zk.desktop.nocache="true".

ID Generator

Each ZUL page we request by Ajax as described above will be an independent desktop. It means the browser window will have several desktops, if we assemble UI this way. Thus, the component's UUID must be unique across different desktops (of the same session^[1]). The default ID generator can handle it well.

However, if you use a customized IdGenerator (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/IdGenerator.html#>), you have to generate component's UUID (org.zkoss.zk.ui.Component) IdGenerator.nextComponentUuid(org.zkoss.zk.ui.Desktop, org.zkoss.zk.ui.Component) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/IdGenerator.html#nextComponentUuid\(org.zkoss.zk.ui.Desktop,\)](#)) correctly. A typical trick is to encode desktop's ID as part of component's UUID.

[1] In short, component's UUID must be unique in the same session. It is OK to be duplicated in different session.

Communicate among ZUL pages

If a ZUL page is loaded separately with Ajax, an independent desktop is created. For example, the following HTML page will create three desktops.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Pragma" content="no-cache" />
<meta http-equiv="Expires" content="-1" />
<script type="text/javascript"
src="http://code.jquery.com/jquery-1.4.2.js"></script>

<title>Assembling at the client with Ajax</title>
</head>
<body>
<table>
<tr>
<td id="top" colspan="2">top</td>
</tr>
<tr>
<td id="left">left</td>
<td id="right">right</td>
</tr>
</table>
<script>
```

```
$(function() {
    $.get("/frags/banner.zul",
        {width : "600px"}, 
        function(response) {
            $("#top").html(response);
        }
    );
    $.get("/frags/leftside.zul",
        {width : "300px"}, 
        function(response) {
            $("#left").html(response);
        }
    );
    $.get("/frags/rightside.zul",
        {width : "300px"}, 
        function(response) {
            $("#right").html(response);
        }
    );
});
```

</script>

</body>

</html>

Since they are in different desktops, you have to use the *group-scoped* event queue^[1] if you want to send events from one desktop (such as leftside.zul) to another (such as rightside.zul). For more information, please refer to Event Queues.

[1] The group-scoped event queue is available only in ZK EE. For ZK CE, you have to use the session-scoped event queue.

Version History

Version	Date	Content
5.0.5	October, 2010	ZUL page is able to be generated as a HTML fragment.

Middleware Layer

Middleware usually means a special software between applications and an operating system. Here, "middleware layer" denotes those frameworks that glue presentation and persistence layer together, including dependency injection frameworks. In the following sections, we will discuss how to integrate them with ZK.

Spring

Overview

Spring Framework is a popular application development framework for enterprise Java. One key element is its infrastructural support: a light-weighted container that manages POJOs as Spring beans and maintain beans' dependency injection relationship. We will talk about several integration ways including wiring and accessing beans in various conditions. We assume that readers have knowledge in Spring's basic configuration and concept such as bean scope, we will therefore not cover these topics here. Please refer to Spring documentation ^[1].

Configuration

The minimal Maven dependency you need is :

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${spring.version}</version>
</dependency>
```

(i) Note: If you don't use Maven, please refer to Spring Framework Reference Documentation to know which JAR file you need.

To integrate ZK application with Spring, the minimal configuration you have to setup is the following:

Spring related configuration in web.xml

```
<!-- Loads the Spring application context configuration -->
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<!-- For using web scoped bean -->
<listener>
    <listener-class>org.springframework.web.context.request.RequestContextListener</listener-class>
</listener>
```

You can enable Spring's classpath scanning to register beans.

WEB-INF/applicationContext.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-3.0.xsd">

    <!-- Scans for application @Components to deploy -->
    <context:component-scan base-package="org.zkoss.reference.developer" />

</beans>
```

- Line 12: Apply `@Component` on those classes that you plan to register them as Spring beans and specify the base package of those classes.

Access a Spring Bean in a ZUL

ZUL provides a feature called variable resolver that allows users to access Spring bean using EL expressions. Simply put the below directive on top of a ZUML page:

```
<?variable-resolver class="org.zkoss.zkplus.spring.DelegatingVariableResolver"
?>
```

Then, on the rest of your page, you can access a Spring-Managed bean directly using its **bean id**.

Assume that we have 2 beans:

```

@Component
@Scope("session")
public class UserPreference {
    ...
}
```

- User preference should be distinct for each user but shared among multiple requests. It is suitable to be a session-scoped bean.

```

@Component
public class SystemConfiguration {
    ...
}
```

- As system configuration should be shared within the whole application, this should be a singleton bean.

Access Spring beans with EL

```
<?variable-resolver class="org.zkoss.zkplus.spring.DelegatingVariableResolver"?>
<window title="Access Bean with different scopes" border="normal" width="700px"
```

```

apply="org.zkoss.reference.developer.composer.ResolverComposer">
<vlayout>
  <hlayout>
    User Preference :
    <label id="sessionValue">${userPreference.value}</label>
  </hlayout>
  <hlayout>
    System Configuration :
    <label id="singletonValue">${systemConfiguration.value}</label>
  </hlayout>
</vlayout>
</window>

```

- The delegating variable-resolver will look-up the bean named `userPreference` automatically for you.

Wire a Spring Bean

It is a very common requirement that we need a Spring bean in a composer, e.g. calling an authentication bean to do login. You can inject Spring beans into ZK Composer/ViewModel in the following ways.

Spring's @Autowire

ZK stores a Composer/ViewModel as a component's attribute, and you can dynamically add/remove components at any time. If you want a fresh new state of a component when each time you add/create it, you should declare a Composer/ViewModel in `prototype` scoped. Then you can use `@Autowire` to wire Spring beans into a Composer/ViewModel.

Wire a Spring bean in a Composer

Alternatively, ZK provides another way to wire a Spring bean to a composer which is not a Spring-managed bean. When we apply a `SelectorComposer` to a ZUL with `org.zkoss.zkplus.spring.DelegatingVariableResolver` mentioned in the previous section, we can apply annotation, `@WireVariable` on a variable we want to wire a Spring bean with. ZK will then wire the corresponding Spring bean on the variable using a variable name that's the same as the bean's name. Or, you can specify the bean's name with `@WireVariable("beanName")`.

A composer that wires Spring beans

```

public class ResolverComposer extends SelectorComposer<Window> {

  @WireVariable
  private OrderService orderService;

  @Wire("#number")
  private Label label;

  @Override
  public void doAfterCompose(Window comp) throws Exception {
    super.doAfterCompose(comp);
  }
}

```

```
label.setValue(Integer.toString(orderService.list().size()));
}

}
```

- Line 1: @WireVariable("beanName") only works inside a SelectorComposer.

A ZUL with Spring variable resolver

```
<?variable-resolver class="org.zkoss.zkplus.spring.DelegatingVariableResolver"?>
<window title="Access Bean with different scopes" border="normal" width="700px"
apply="org.zkoss.reference.developer.spring.composer.ResolverComposer">
...
</window>
```

Wire a Spring bean in a ViewModel

Wiring a Spring bean in a ViewModel is very similar to the case in a composer, simply apply @WireVariable with variable resolver. In the example below we put variable resolver in a zul with a directive.

A ViewModel that wires a Spring bean

```
public class OrderVM {

    @WireVariable
    private OrderService orderService;

    ...
}
```

The zul uses OrderVM with a Spring variable resolver

```
<?variable-resolver class="org.zkoss.zkplus.spring.DelegatingVariableResolver"?>
<zk>
<window title="Order Management" border="normal" width="600px" apply="org.zkoss.bind.BindComposer"
viewModel="@id('vm')"
@init('org.zkoss.reference.developer.spring.order.viewmodel.OrderVM')
validationMessages="@id('vmsgs')">

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

Adding Variable Resolver to a Composer/ViewModel

Adding a variable resolver to a ZUL will make it available to all composers on the ZUL. If you want to add a variable resolver to a specific composer (or ViewModel) only, you should apply the annotation

`@VariableResolver(org.zkoss.zkplus.spring.DelegatingVariableResolver.class)`
on the class that inherits `SelectorComposer` or a `ViewModel`. Then, apply `@WireVariable` on variables like we did in the previous section.

Example code are as follows:

```
@VariableResolver(org.zkoss.zkplus.spring.DelegatingVariableResolver.class)
public class SpringComposer extends SelectorComposer<Window> {

    @WireVariable
    private OrderService orderService;
    ...
}
```

Retrieve a Spring Bean Programmatically

`org.zkoss.zkplus.spring.SpringUtil` is a utility class that allows you to get Spring-managed beans by their name in Java.

```
public class SpringComposer extends SelectorComposer<Window> {

    @Wire("#number")
    private Label label;

    @Override
    public void doAfterCompose(Window comp) throws Exception {
        super.doAfterCompose(comp);
        OrderService orderService =
            (OrderService) SpringUtil.getBean("orderService");

        label.setValue(Integer.toString(orderService.list().size()));
    }
}
```

Integrate Spring Webflow and Security

ZK also provides integration to other Spring projects such as Spring Security and Spring Webflow with ZK Spring. Please refer to ZK Spring Essentials for details.

Example Source Code

You can get all source code mentioned in this chapter at GitHub/zkoss/zkbooks^[2].

References

[1] <http://www.springsource.org/spring-framework#documentation>

[2] <https://github.com/zkoss/zkbooks/tree/master/developersreference/integration.spring>

CDI

Overview

Contexts and Dependency Injection (CDI) is one of Java EE 6 features and is composed of a set of services designed for using with stateful objects. It also allows developers to integrate various kinds of objects in a loosely coupled and type safe way. We will talk about several ways of integration including injecting and accessing CDI beans under different conditions. We assume that readers have knowledge in basic CDI configuration and concept such as bean scope, we will therefore not cover those topics here. Please refer to Oracle's CDI tutorial^[1].

Access a CDI Bean in a ZUL

ZUL provides a feature called variable resolver that allows users to access CDI bean using EL expression. To do this, simply put the below directive on top of a ZUML page:

```
<?variable-resolver class="org.zkoss.zkplus.cdi.DelegatingVariableResolver" ?>
```

Then, in the rest of your page, you can access a CDI bean which has @Named directly using its **bean EL name**.

Session scoped bean

```
@SessionScoped @Named  
public class UserPreference implements Serializable{  
    ...  
}
```

- User preference should be distinct for each user but shared among multiple requests. It is suitable to be a session scoped bean.

Application scoped bean

```
@ApplicationScoped @Named  
public class SystemConfiguration implements Serializable{  
    ...  
}
```

- As system configuration should be shared within the whole application, it should be an application scoped bean.

Access bean using EL in a ZUL

```

<?variable-resolver class="org.zkoss.zkplus.cdi.DelegatingVariableResolver"?>
...
<hlayout>
    User Preference :
    <label id="sessionValue">${userPreference.value}</label>
</hlayout>
<hlayout>
    System Configuration :
    <label id="applicationValue">${systemConfiguration.value}</label>
</hlayout>
...

```

Wire CDI beans

Wire a CDI bean in a Composer

It is likely that we need to use a CDI bean in a composer, for example calling a service layer object to perform business logic. If a composer is a CDI bean, we can simply use `@Inject` to inject all collaborators. However, we do not recommend this approach as explained in previously.

ZK provides another approach to wire a CDI bean in a composer which is not a CDI bean. With help of `org.zkoss.zkplus.spring.DelegatingVariableResolver` and `@WireVariable`, we can inject CDI beans into a composer. There are two ways to apply a variable resolver to a composer. We can

1. Put it in a zul with directive mentioned in the previous section or,
2. In a Java class with annotation, `@VariableResolver` then apply the annotation, `@WireVariable` on the variables which we want to inject CDI beans to.

Example code are as follow:

A composer injected with a CDI bean

```

public class ResolverComposer extends SelectorComposer<Window> {

    @WireVariable("normalOrderService")
    NormalOrderService orderService;

    @Wire("#number")
    private Label label;

    @Override
    public void doAfterCompose(Window comp) throws Exception {
        super.doAfterCompose(comp);

        label.setValue(Integer.toString(orderService.findAll().size()));
    }
}

```

A ZUL with CDI variable resolver

```
<?variable-resolver class="org.zkoss.zkplus.spring.DelegatingVariableResolver"?>
<window title="Access Bean with different scopes" border="normal" width="700px"
apply="org.zkoss.reference.developer.spring.composer.ResolverComposer">
...
</window>
```

Wire a CDI bean in a ViewModel

Like wiring in a composer, we apply CDI variable resolver with directive and `@WireVariable` to inject CDI beans.

```
public class MyViewModel {

    @WireVariable
    private UserPreference userPreference;
    @WireVariable
    private ProductService productService;

    private List<String> productList;

    @Init
    public void doAfterCompose(Window comp) throws Exception {
        productList = productService.findAll();
    }

    public UserPreference getUserPreference() {
        return userPreference;
    }

}

<?variable-resolver class="org.zkoss.zkplus.cdi.DelegatingVariableResolver"?>
<window border="normal" width="500px"
apply="org.zkoss.bind.BindComposer"

viewModel="@id('vm')@init('org.zkoss.reference.developer.composer.MyViewModel')">
...
</window>
```

Adding Variable Resolver to a Composer (or ViewModel)

Adding a variable resolver to a ZUL will make it available to all composers on the ZUL. If you only want to add a variable resolver to a specific composer (or ViewModel), you should apply the annotation

`@VariableResolver(org.zkoss.zkplus.cdi.DelegatingVariableResolver.class)`

on the class that inherits `SelectorComposer` or a `ViewModel`, then, apply `@WireVariable` on variables like shown in the previous section.

Example code are as follows:

```
@VariableResolver(org.zkoss.zkplus.cdi.DelegatingVariableResolver.class)
public class MyComposer extends SelectorComposer<Window> {

    @WireVariable
    private UserPreference userPreference;
    ...
}
```

Warning: Declare a Composer (or ViewModel) as a CDI bean

Developers might tend to make a composer (or a ViewModel) as a CDI bean, but we don't recommend this approach. Because none of CDI's scopes matches correctly with the life cycle of the composers. The scope of a composer is "desktop" scope. It is shorter than "session" and longer than "prototype". Only ZK knows when to create composers (or ViewModel), so it's better to let composers be managed by ZK.

If you insist on making composers (or ViewModel) as CDI beans, `@Dependent` scope could be a feasible scope but you need to use with care; each time you try to resolve a composer bean, you will get a new instance of a composer. If the composer stores some states, it would cause inconsistency states among multiple composers.

Example Source Code

All source code of examples used in this chapter can be found here ^[2].

Version History

Version	Date	Content
6.5.0	November 2012	Rewrite for improvement.

References

[1] <http://docs.oracle.com/javaee/6/tutorial/doc/gjbnr.html>

[2] <https://github.com/zkoss/zkbooks/tree/master/developersreference/integration.cdi>

EJB

Enterprise JavaBeans (EJB) technology is the server-side component architecture for Java EE. Here we describe how to use it in a ZUML document.

Here we use JBoss [1] as the example. The configuration of the server might vary from one server to another, but the ZUML document is the same.

Notice that if you would like to access EJB in Java (such as in a composer or in a richlet), you could skip this section (since you could use the approach described in any EJB guide).

Use JndiVariableResolver to Resolve EJB in EL Expressions

Referencing an EJB in an EL expression is straightforward: specifying JndiVariableResolver [2] in the variable-resolver directive. For example,

```
<?variable-resolver class="org.zkoss.zkplus.jndi.JndiVariableResolver" ?>
<window>
...
</window>
```

Depending your configuration, you might have to pass extra information about JNDI to it such as:

```
<?variable-resolver class="org.zkoss.zkplus.jndi.JndiVariableResolver"
    arg0="ZkEJB3Demo"
    arg1="mail=java:comp/env/mail,sec=java:comp/security/module" ?>
<!--
    arg0: prepend - the prepended part of JDNDI name
    arg1: mapping - the key-value pairs for JNDI names and the
corresponding variable names
-->
<window>
...
</window>
```

JndiVariableResolver [2] will resolve variables in the following order:

1. java:comp/env
2. java:comp
3. java:
4. Variable could be found as a session beans with the `prepend` argument (`arg0`).
5. The key-value pairs which is defined in the `mapping` argument (`arg1`)

Example: Retrieve Session Beans

The session beans are bound to the `java:comp/env` configured by `jboss-web.xml` and `web.xml`. For example, suppose we have them as follows:

`jboss-web.xml`:

```
<ejb-local-ref>
    <ejb-ref-name>personLocalBean</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <local>demo.PersonBeanLocal</local>
    <local-jndi-name>ZkEJB3Demo/PersonBean/local</local-jndi-name>
</ejb-local-ref>
```

`web.xml`:

```
<ejb-local-ref>
    <ejb-ref-name>personLocalBean</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <local-home>demo.PersonBeanLocal</local-home>
    <local>demo.PersonBeanLocal</local>
</ejb-local-ref>
```

Then, we could access them as follows.

```
<?variable-resolver class="org.zkoss.zkplus.jndi.JndiVariableResolver" ?>
<listbox width="600px">
    <listhead sizable="true">
        <listheader label="name" sort="auto"/>
        <listheader label="email" sort="auto"/>
    </listhead>
    <listitem forEach="${personLocalBean.allPersons}"> <!-- resolve personLocalBean from JNDI -->
        <listcell label="${each.name}" />
        <listcell label="${each.email}" />
    </listitem>
</listbox>
```

The variables provided by a variable resolver is also available to the Java code in zscript. For example,

```
<zscript>
personLocalBean.createDemoData();
</zscript>
```

Example: Retrieve EntityManagerFactory

Persistence units are not bound into JNDI by default, so we have to define JBoss specific properties in `persistence.xml` to bind them into JNDI. For example,

```
</persistence-unit>
<properties>
    <property name="jboss.entity.manager.factory.jndi.name" value="java:comp/entityManagerFactory"/>
</properties>
</persistence-unit>
```

Then, we could retrieve the entity manager factory by use of JndiVariableResolver^[2].

Source Code

You can get all source code mentioned in this section at github^[3]

References

[1] <http://jboss.org>

[2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkplus/jndi/JndiVariableResolver.html#>

[3] <https://github.com/zkoss/zkbooks/tree/master/developersreference/integration.ejb>

Persistence Layer

In the following sections, we are going to describe considerations and issues when you use persistence frameworks with ZK.

JDBC

ZK aims to be as thin as the presentation tier. In addition, as the code executes at the server, so connecting database is no different from any desktop applications. In other words, ZK doesn't change the way you access the database, no matter you use JDBC or other persistence framework, such as Hibernate^[1].

Use JDBC

The simplest way to use JDBC, like any JDBC tutorial might suggest, is to use `java.sql.DriverManager`. Here is an example to store the name and email into a MySQL^[2] database.

```
public class JdbcComposer extends SelectorComposer<Window> {

    private static Logger log =
Logger.getLogger(JdbcComposer.class.getName());
    @Wire
    private Textbox name;
    @Wire
    private Textbox email;

    @Listen("onClick = button")
    public void submit() {
        PreparedStatement stmt = null;
        Connection conn = null;
        try {
            //load driver and get a database connection
            Class.forName("com.mysql.jdbc.Driver");
            conn = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/test",
                "root",
                "password"
            );
            stmt = conn.prepareStatement("insert into user values(?, ?)");
            stmt.setString(1, name.getValue());
            stmt.setString(2, email.getValue());
            stmt.executeUpdate();
        } catch (Exception e) {
            log.error(e.getMessage());
        }
    }
}
```

```
"jdbc:mysql://localhost/test?user=root&password=R3f@ct0r");
        stmt = conn.prepareStatement("INSERT INTO user
values(?, ?)");

        //insert what end user entered into database table
        stmt.setString(1, name.getValue());
        stmt.setString(2, email.getValue());

        //execute the statement
        stmt.executeUpdate();
    } catch(Exception e) {
        log.severe(e.toString());
    } finally { //cleanup
        if (stmt != null) {
            try {
                stmt.close();
            } catch (SQLException ex) {
                log.severe(ex.toString()); //log and
ignore
            }
        }
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException ex) {
                log.severe(ex.toString()); //log and
ignore
            }
        }
    }
}
```

```
<window title="JDBC demo" border="normal" apply="org.zkoss.reference.developer.integration.JdbcComposer">

    <vbox>

        <hbox>Name : <textbox id="name"/></hbox>
        <hbox>Email: <textbox id="email"/></hbox>
        <button label="submit"/>
    </vbox>

</window>
```

Though this way is simple, but it has obvious drawback. After all, ZK applications are web-based applications, where loading is unpredictable and treasures resources such as database connections have to be managed more effectively.

Luckily, all J2EE frameworks and Web servers support a utility called connection pooling. It is straightforward to use, while managing the database connections well. We will discuss more in the next section.

Tip: Unlike other Web applications, it is possible to use DriverManager with ZK, though *not recommended*.

First, you could cache the connection in the desktop, reuse it for each event, and close it when the desktop becomes invalid. It works just like traditional Client/Server applications. Like Client/Server applications, it works efficiently only if there are at most tens concurrent users.

To know when a desktop becomes invalid, you have to implement a listener by use of DesktopCleanup^[3].

Use a Connection Pool

Connection pool is a mechanism for creating and managing a pool of connections that are ready to use by a thread that needs them. Instead of closing a connection immediately, it keeps them in a pool such that the next connection request could reuse them. Connection pool, in addition, has a lot of benefits, such as control resource usage.

It's recommended to use connection pool if you want to operate Java Connection directly when developing web-based applications, including ZK applications.

The usage of connection pool is simple: configure, connect and close. The way to connect and close a connection is very similar to the ad-hoc approach, while the configuration depends on what web server and database server are in use.

Connect and Close a Connection

After configuring the connection pool (which will be discussed in the following section), you could use JNDI to retrieve an connection as follows.

```
public class DatasourceComposer extends SelectorComposer<Window> {

    @Wire
    private Textbox name;
    @Wire
    private Textbox email;

    @Listen("onClick = button")
    public void submit() {

        Connection conn = null;
        PreparedStatement stmt = null;
        try {
            DataSource ds = (DataSource) new
InitialContext().lookup("java:comp/env/jdbc/MyDB");
            conn = ds.getConnection();
            //remember that we specify autocommit as false in the
context.xml
            conn.setAutoCommit(true);
            stmt = conn.prepareStatement("INSERT INTO user
values(?, ?)");
            stmt.setString(1, name.getValue());
            stmt.setString(2, email.getValue());
            stmt.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

```
        stmt.close();
        stmt = null;
    } catch (SQLException e) {
        try{
            conn.rollback();
        }catch(SQLException ex){
            //log
        }
        //(optional log and) ignore
    } catch (Exception e) {
        //log
    } finally { //cleanup
        if (stmt != null) {
            try {
                stmt.close();
            } catch (SQLException ex) {
                //(optional log and) ignore
            }
        }
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException ex) {
                //(optional log and) ignore
            }
        }
    }
}

<window title="JDBC demo" border="normal" apply="org.zkoss.reference.developer.integration.DatasourceComposer">
    <vbox>
        <hbox>Name : <textbox id="name"/></hbox>
        <hbox>Email: <textbox id="email"/></hbox>
        <button label="submit"/>
    </vbox>
</window>
```

Notes:

- It is important to close the statement and connection after use.
- You could access multiple databases at the same time with multiple connections. Depending on the configuration and J2EE/Web servers, these connections could even form a distributed transaction.

Configure Connection Pool

The configuration of connection pool varies from one J2EE/Web/Database server to another. Here we illustrate some of them. You have to consult the document of the server you are using.

Tomcat 5.5 (and above) + MySQL

To configure connection pool for Tomcat 5.5, you have to edit `$TOMCAT_DIR/conf/context.xml`^[4], and add the following content under the `<Context>` element. The information that depends on your installation and usually need to be changed is marked in the blue color.

```

<!-- The name you used above, must match _exactly_ here!
The connection pool will be bound into JNDI with the name
"java:/comp/env/jdbc/MyDB"

-->
<Resource name="jdbc/MyDB" username="someuser" password="somepass"
url="jdbc:mysql://localhost:3306/test"
auth="Container" defaultAutoCommit="false"
driverClassName="com.mysql.jdbc.Driver" maxActive="20"
timeBetweenEvictionRunsMillis="60000"
type="javax.sql.DataSource" />

```

Then, in `web.xml`, you have to add the following content under the `<web-app>` element as follows.

```

<resource-ref>
  <res-ref-name>jdbc/MyDB</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>

```

Notes

- [1] <http://www.hibernate.org/>
- [2] <http://www.mysql.com/>
- [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/DesktopCleanup.html#>
- [4] Thanks Thomas Muller (<http://asconet.org:8000/antville/oberinspector>) for correction.

See also (<http://tomcat.apache.org/tomcat-5.5-doc/jndi-resources-howto.html>) and (http://en.wikibooks.org/wiki/ZK/How-Tos/HowToHandleHibernateSessions#Working_with_the_Hibernate_session) for more details.

JBoss + MySQL

The following instructions is based on section 23.3.4.3 of the reference manual of MySQL 5.0.

To configure connection pool for JBoss, you have to add a new file to the directory called `deploy` (`$JBOSS_DIR/server/default/deploy`). The file name must end with `"*-ds.xml"` (* means the database, please refer to `$JBOSS_DIR/docs/examples/jca/`), which tells JBoss to deploy this file as JDBC Datasource. The file must have the following contents. The information that depends on your installation and usually need to be changed is marked in the blue color.

`mysql-ds.xml`:

```

<datasources>
  <local-tx-datasource>
    <!-- This connection pool will be bound into JNDI with the name
    "java:/MyDB" -->

```

```

<jndi-name>MyDB</jndi-name>

<connection-url>jdbc:mysql://localhost:3306/test</connection-url>

<driver-class>com.mysql.jdbc.Driver</driver-class>

<user-name>someuser</user-name>

<password>somepass</password>

<min-pool-size>5</min-pool-size>

<!-- Don't set this any higher than max_connections on your
MySQL server, usually this should be a 10 or a few 10's
of connections, not hundreds or thousands --&gt;

&lt;max-pool-size&gt;20&lt;/max-pool-size&gt;

<!-- Don't allow connections to hang out idle too long,
never longer than what wait_timeout is set to on the
server...A few minutes is usually okay here,
it depends on your application
and how much spiky load it will see --&gt;

&lt;idle-timeout-minutes&gt;5&lt;/idle-timeout-minutes&gt;

<!-- If you're using Connector/J 3.1.8 or newer, you can use
our implementation of these to increase the robustness
of the connection pool. --&gt;

&lt;exception-sorter-class-name&gt;com.mysql.jdbc.integration.jboss.ExtendedMysqlExceptionSorter&lt;/exception-sorter-class-name&gt;

&lt;valid-connection-checker-class-name&gt;com.mysql.jdbc.integration.jboss.MysqlValidConnectionChecker&lt;/valid-connection-checker-class-name&gt;

&lt;/local-tx-datasource&gt;

&lt;/datasources&gt;
</pre>

```

To specify the JNDI name at which the datasource is available , you have to add a `jboss-web.xml` file under the WEB-INF folder.

`jboss-web.xml`

```

<jboss-web>
<resource-ref>
    <res-ref-name>jdbc/MyDB</res-ref-name>
    <jndi-name> java:/MyDB </jndi-name>
</resource-ref>
</jboss-web>

```

In `web.xml`, you have to add the following content under the `<web-app>` element as follows.

```

<resource-ref>
    <res-ref-name>jdbc/MyDB</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>

```

```
</resource-ref>
```

JBoss + PostgreSQL

```
<datasources>
  <local-tx-datasource>
    <!-- This connection pool will be bound into JNDI with the name
        "java:/MyDB" -->
    <jndi-name>MyDB</jndi-name>

    <!-- jdbc:postgresql://[servername]:[port]/[database name] -->
    <connection-url>jdbc:postgresql://localhost/test</connection-url>

    <driver-class>org.postgresql.Driver</driver-class>
    <user-name>someuser</user-name>
    <password>somempass</password>
    <min-pool-size>5</min-pool-size>
    <max-pool-size>20</max-pool-size>
    <track-statements>false</track-statements>
  </local-tx-datasource>
</datasources>
```

download

- Please download the source(Tomcat 5.5 (and above) + MySQL) ([https://sourceforge.net/projects/zkforge/files/Small Talks/JDBC\(JNDI sample\)/Mysql_tomcat.war/download](https://sourceforge.net/projects/zkforge/files/Small%20Talks/JDBC(JNDI%20sample)/Mysql_tomcat.war/download))
- Please download the source(JBoss + MySQL) ([https://sourceforge.net/projects/zkforge/files/Small Talks/JDBC\(JNDI sample\)/jboss+mysql.zip/download](https://sourceforge.net/projects/zkforge/files/Small%20Talks/JDBC(JNDI%20sample)/jboss+mysql.zip/download))

Hibernate

Overview

Due to *object/relational paradigm mismatch* ^[1], developers tend to use ORM (*object/relational mapping*) framework to convert object-oriented model to relational model and vice versa. *Hibernate* is the most popular ORM framework in Java world. We will talk about some integration topics in this chapter such as lazy initialization with Spring. If you haven't read about basic concepts and installation of Hibernate, please refer to *Hibernate Documentation* ^[2]. The example we give in this chapter is based on **Hibernate 4.0.0.final** and **Spring 3.1.2.RELEASE**.

Integrate with Different DAO implementation

The *Data Access Object (DAO)* pattern is a good practice to implement a persistence layer. This pattern encapsulates data access codes written by Hibernate API from business tier. A DAO object exposes an interface to business object and performs persistence operation relating to a particular persistent entity.

According to *Hibernate Reference Manual's* suggestion ^[3], we should apply *session-per-request* pattern to manage sessions and transactions. This pattern needs an interceptor to open a contextual session with a transaction when a request is going to be handled and close the session before respond is sent to client (aka. *open session in view* pattern). A common implementation for page-based application is a servlet filter.

Applying this pattern also solves a common "LazyInitializationException" problem that most developers encounter when using lazy fetching strategy. In brief, Hibernate session is usually closed after a DAO object has performed an operation (a.k.a *session-per-operation* pattern). Those persistent objects become *detached* after the associated sessions are closed. If we access a detached object's lazy-loaded collection when rendering the view. We will get an error message like `LazyInitializationException: no session or session was closed`. For detailed explanation, please refer to the article "Open Session in View" on *Hibernate community*^[4]. As we apply *session-per-request* pattern, a Hibernate session is kept open when a View is accessing lazy-loaded collection. Those objects queried by the Hibernate session becomes detached later (after the interceptor closes the Hibernate session), so the previously mentioned problem is resolved.

Homemade DAO

Here we introduce how to implement an DAO without other frameworks (e.g. Spring).

Configuration

The minimal Maven dependency you need is:

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>4.0.0.Final</version>
    <scope>compile</scope>
</dependency>
```

 **Note:** If you don't use Maven, please refer to *Hibernate Reference Documentation* to know which JAR file you need.

Utility Class

A simple way to implement a DAO is to control Hibernate sessions and transactions manually, hence we need a utility class to get SessionFactory. ZK's HibernateUtil^[5] has been deprecated since 6.0.2, you can write your own one according to the code example in Hibernate Reference Manual.^[6] Here we provide a simple example.

Utility class to get SessionFactory

```
package org.zkoss.reference.developer.hibernate.dao;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
    private static SessionFactory sessionFactory;
    static {
        try {
            sessionFactory = new Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

An Open Session Listener

For *open session in view* pattern, we need an interceptor to open sessions. In ZK, we need to intercept all requests including AU requests, so we implement ZK's Life Cycle Listener to achieve this. (Our listener's implementation is based on the filter mentioned by a Hibernate article "Open Session in View".^[7]) This listener opens a session and begins a transaction at the beginning of an execution (ZK's HTTP request wrapper, then commits (or rollback) at the end of the execution.

Extracted from OpenSessionInViewListener

```
public class OpenSessionInViewListener implements ExecutionInit,
ExecutionCleanup {
    private static final Log log =
    Log.lookup(OpenSessionInViewListener.class);

    public void init(Execution exec, Execution parent) {
        if (parent == null) { //the root execution of a servlet
request
            log.debug("Starting a database transaction: "+exec);
            HibernateUtil.getSessionFactory().getCurrentSession().beginTransaction();
        }
    }
}
```

```

        }

    }

    public void cleanup(Execution exec, Execution parent, List errs)
    {
        if (parent == null) { //the root execution of a servlet
request
            if (errs == null || errs.isEmpty()) {
                log.debug("Committing the database transaction:
"+exec);

HibernateUtil.getSessionFactory().getCurrentSession().getTransaction().commit();
            } else {
                final Throwable ex = (Throwable) errs.get(0);
                rollback(exec, ex);
            }
        }
    }

    private void rollback(Execution exec, Throwable ex) {
        try {
            if
(HibernateUtil.getSessionFactory().getCurrentSession().getTransaction().isActive())
{
            log.debug("Trying to rollback database
transaction after exception:"+ex);

HibernateUtil.getSessionFactory().getCurrentSession().getTransaction().rollback();
        }
        catch (Throwable rbEx) {
            log.error("Could not rollback transaction after
exception! Original Exception:\n"+ex, rbEx);
        }
    }
}

```

- Line 7: Call getCurrentSession() to get a contextual session. [8]

Add configuration in zk.xml to make the listener work.

Configure listener in zk.xml

```

<zk>
    <listener>

        <listener-class>org.zkoss.reference.developer.hibernate.web.OpenSessionInViewListener</listener-class>
    </listener>
</zk>

```

DAO Implementation

The listener begins and commits transactions keeping DAO's implementation simple. Just use the utility class to get current Hibernate session to perform the operation.

Simple DAO implementation

```
public class OrderDao {  
  
    public List<Order> findAll() {  
        Session session =  
        HibernateUtil.getSessionFactory().getCurrentSession();  
        Query query = session.createQuery("select o from Order as  
o");  
        List<Order> result = query.list();  
        return result;  
    }  
  
    /**  
     * rollback is handled in filter.  
     * @param newOrder  
     * @return  
     * @throws HibernateException  
     */  
    public Order save(Order newOrder) throws HibernateException{  
        Session session =  
        HibernateUtil.getSessionFactory().getCurrentSession();  
        session.save(newOrder);  
        session.flush();  
        return newOrder;  
    }  
}
```

Finally, we can use the DAO class in a ViewModel to manipulate domain objects.

Use DAO in a ViewModel

```
public class OrderViewModel {  
  
    private OrderDao orderDao = new OrderDao();  
  
    private List<Order> orders ;  
    private Order selectedItem;  
  
    @Init  
    public void init(){  
        orders = orderDao.findAll();  
        if (!orders.isEmpty()){  
            setSelectedItem(orders.get(0));  
        }  
    }  
}
```

```
//omit setter and getter for brevity  
}
```

Spring-based DAO

With Spring provided dependency injection and ORM support, here our efforts are simplified quite substantially. We'll demonstrate one typical usage in non-managed environment. To apply *session-per-request* pattern, we can use Spring provided `OpenSessionInViewFilter` instead of writing our own one. According to the suggestion in Spring Reference Documentation 3.1, using plain Hibernate API to implement a DAO is the current recommended usage pattern. In the DAO, we can also easily retrieve `SessionFactory` by dependency inject without any utility classes (`HibernateUtil`). Besides, declarative transaction management and rollback rule also reduces our work. The following are the related code snippets.

Configuration

The minimal dependencies you need are Hibernate-Core, Spring-Web, Spring-ORM, and Cglib:

```
<dependency>  
    <groupId>org.hibernate</groupId>  
    <artifactId>hibernate-core</artifactId>  
    <version>4.0.0.Final</version>  
</dependency>  
<dependency>  
    <groupId>org.springframework</groupId>  
    <artifactId>spring-web</artifactId>  
    <version>3.1.2.RELEASE</version>  
</dependency>  
<dependency>  
    <groupId>org.springframework</groupId>  
    <artifactId>spring-orm</artifactId>  
    <version>3.1.2.RELEASE</version>  
</dependency>  
<dependency>  
    <groupId>cglib</groupId>  
    <artifactId>cglib</artifactId>  
    <version>2.2</version>  
</dependency>
```

We use basic configuration for Spring.

Spring configuration for Hibernate

```
<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
       xmlns:context="http://www.springframework.org/schema/context"  
       xmlns:tx="http://www.springframework.org/schema/tx"  
       xsi:schemaLocation="  
           http://www.springframework.org/schema/beans  
  
           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
```

```
http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.0.xsd
http://www.springframework.org/schema/tx

http://www.springframework.org/schema/tx/spring-tx-3.0.xsd">

<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource" >
    <property name="driverClassName" value="org.hsqldb.jdbcDriver" />
    <property name="url" value="jdbc:hsqldb:file:data/store" />
    <property name="username" value="sa" />
    <property name="password" value="" />
</bean>
<!--
    hibernate.current_session_context_class=thread
-->
<bean id="sessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="hibernateProperties">
        <value>
            hibernate.dialect=org.hibernate.dialect.HSQLDialect
                hibernate.hbm2ddl.auto=crate
                hibernate.show_sql=true
                hibernate.connection.pool_size=5
                hibernate.connection.autocommit=false
            </value>
        </property>
        <property name="annotatedClasses">
            <list>
                <value>org.zkoss.reference.developer.hibernate.domain.Order</value>
                <value>org.zkoss.reference.developer.hibernate.domain.OrderItem</value>
            </list>
        </property>
    </bean>
    <bean id="transactionManager" class="org.springframework.orm.hibernate4.HibernateTransactionManager">
        <property name="sessionFactory" ref="sessionFactory" />
    </bean>
<tx:annotation-driven />
<!-- Scans for application @Components to deploy -->
<context:component-scan base-package="org.zkoss.reference.developer.hibernate" />
</beans>
```

- Line 40: For Hibernate 3.x, some package names should be changed to
org.springframework.orm.hibernate3.* , e.g.
org.springframework.orm.hibernate3.HibernateTransactionManager.

OpenSessionInViewFilter

Spring already provides a **OpenSessionInViewFilter** to solve lazy loading in web views problems. This filter makes Hibernate Sessions available via the current thread, which will be auto-detected by Spring's transaction managers. For detailed description and usage, please refer to Spring's Javadoc.

Configure OpenSessionInViewFilter in web.xml

```
<filter>
    <filter-name>OpenSessionInViewFilter</filter-name>

    <filter-class>org.springframework.orm.hibernate4.support.OpenSessionInViewFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>OpenSessionInViewFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

DAO Implementation

For a Spring-powered DAO, we can use injected SessionFactory and @Transactional to perform persistence operation.

DAO empowered by Spring

```
@Repository
public class SpringOrderDao {

    @Autowired
    private SessionFactory sessionFactory;

    @Transactional(readOnly=true)
    public List<Order> queryAll() {
        Session session = sessionFactory.getCurrentSession();
        Query query = session.createQuery("select o from Order as o");
        List<Order> result = query.list();
        return result;
    }

    @Transactional
    public Order save(Order newOrder) {
        Session session = sessionFactory.getCurrentSession();
        session.save(newOrder);
        session.flush();
        return newOrder;
    }
}
```

```
//omit other codes
}
```

To use this Spring-based DAO in a composer (or a ViewModel), ZK provides several ways like variable resolvers. Please refer to [ZK Developer's Reference/Integration/Middleware Layer/Spring](#).

Lazy Initialization Issue among AU Requests

Although we apply *open session in view* pattern to keep a session open when a page is rendered, this makes ZUL access a lazy-loaded collection without errors **when you visit the ZUL at first request**. However, if your event handling methods (or command methods) accesses a lazy-loaded property of a detached object, you will still get a `LazyInitializationException` when a user interacts with the ZUL. This is because even though the filter opens a session for each request, the detached objects don't attach to the session automatically.^[9] The two DAO implementations demonstrated previously both have this problem and there are two solutions for it.

1. set fetching strategy to **eagerly fetch**.
2. Re-query the detached object manually.

If you are not dealing with large amount of data, you can choose the first solution by simply changing your fetching strategy to eager for properties.

We will use an example to demonstrate the second solution. The following is a "Order Viewer" system. The upper *Listbox* contains a list of orders and the lower *Grid* contains the details of items that are selected in the order. One order may contain many order items (one-to-many mapping), and we set order items collection to lazy fetching . When we select an order, the *Grid* displays detailed items of the selected order which means accessing a lazy-loaded collection.

```
@Entity
@Table(name="orders")
public class Order {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String status = PROCESSING;
    private String description;

    @OneToMany(mappedBy="orderId", fetch=FetchType.LAZY)
    private List<OrderItem> items = new ArrayList<OrderItem>();

    //other codes...
}
```

By default, we set the *Listbox* selection on the first row. When ZUL accesses the selected order's lazy-loaded items collection, Hibernate can load it successfully with the help of open-session-in-view filter because session is still open. However, if we click the second row which also accesses a detached `Order` object's items collection, we should re-load the object with Hibernate session or we'll get `LazyInitializationException`.

The screenshot shows a user interface titled "Order Viewer". At the top, there is a table with three columns: "ID", "Description", and "Status". Two rows are visible: row 1 has ID 1, description "nobody at home in day time", and status "processing"; row 2 has ID 2, description "urgent case", and status "processing". A cursor arrow is positioned over the second row. Below the table, there is a section titled "Details" containing another table with three columns: "Name", "Quantity", and "Price". This table has two rows: "Cookies" with quantity 10 and price 4.0, and "Toast" with quantity 5 and price 3.1.

source of above screen [10]

In order to reload a detached object, we pass the selected order to DAO object and reload it.

Reload selected object

```
public class OrderViewModel {
    private OrderDao orderDao = new OrderDao();

    private List<Order> orders;
    private Order selectedItem;

    @Init
    public void init() {
        orders = orderDao.findAll();
        setSelectedItem(orders.get(0));
    }

    public Order getSelectedItem() {
        if (selectedItem!=null) {
            selectedItem = orderDao.reload(selectedItem);
        }
        return selectedItem;
    }

    //omit other methods for brevity
}
```

- Line 11: Initialize the `Listbox` selection with the `Order` object at index 0 of `orders`.
- Line 16: Re-query the `selectedItem`.

We use `Session.load()` to re-query the `Order` object with its id, this newly-retrieved object still has an open Hibernate session. Then when we access the lazy-loaded collection (`items`) in the ZUL, Hibernate can retrieve the collection for us. After doing so, we can eliminate `LazyInitializationException`.

Re-attach to a session

```
public class OrderDao {
    ...
    /**
     * Initialize lazy-loaded collection.
     * @param order
     * @return
     */
    public Order reload(Order order) {
        return
            (Order) HibernateUtil.getSessionFactory().getCurrentSession().load(Order.class, order.getId());
    }
}
```

Lazy Initialization Issue Under Render on Demand

Some AU requests cannot be interfered by developers, such as a "Render On Demand" request where the rendering request is handled implicitly by a component itself. Under this situation, if a component needs to **render some data from a detached object's lazy-loaded collection**, developers won't have a chance to reload detached objects during the rendering to avoid LazyInitializationException.

Assume we have a *Listbox*, it only displays 10 rows by default and it's not in "page" mold. One of its columns display a lazy-loaded collection's size (`each.items.size()`) of an `Order` object.

Listbox that accesses lazy-loaded property

```
<window title="" border="normal" width="600px" apply="org.zkoss.bind.BindComposer"
        viewModel="@id('vm')"

@init('org.zkoss.reference.developer.hibernate.vm.RodViewModel')>
    Contain a customized model that reload lazy-loaded
    collection from database
    <listbox model="@load(vm.orderListModel)" rows="10">
        <listhead>
            <listheader label="ID" width="50px" />
            <listheader label="Description" />
            <listheader label="Item Count" width="150px" />
        </listhead>
        <template name="model">
            <listitem>
                <listcell label="@load(each.id) " />
                <listcell label="@load(each.description)" />
                <listcell label="@load(each.items.size())" />
            </listitem>
        </template>
    </listbox>
    ...

```

If we just pass a Java `List` object to be the model of the `Listbox`, when a user scrolls down to view other rows, ZK will send AU request to retrieve data for those un-rendered rows. `Listbox` will try to access lazy-loaded collection but objects in the list are already detached, and we will get `LazyInitializationException`. During this rendering process, developers will not be notified and cannot interfere this process to reload detached objects. One solution is to **implement a custom `ListModel`**^[7] for the component.

We demonstrate 2 implementations here for your reference. The first one is simpler but less efficient; it re-queries each detached object when a component requests it.

Reloaded ListModel

```
public class OrderListModel extends AbstractListModel<Order> {

    private OrderDao orderDao;
    List<Order> orderList = new LinkedList<Order>();

    public OrderListModel(List<Order> orders, OrderDao orderDao) {
        this.orderList = orders;
        this.orderDao = orderDao;
    }

    @Override
    public Order getElementAt(int index) {
        //throw a runtime exception if orderDao does not find
target object
        Order renewOrder = orderDao.reload(orderList.get(index));
        return renewOrder;
    }

    @Override
    public int getSize() {
        return orderList.size();
    }
}
```

- Line 1: We extends `AbstractListModel`^[6] to build our list model for it handles selection for us, but we have to implement `Order`'s `equals()` and `hashCode()`.
- Line 14: Re-query the detached object by its id and return a persistent one.

The second one is more complicated but more efficient; it re-queries a one page size data each time and stores as a cache in an execution. If the cache has the object that the component requests, it returns the one in cache without re-querying again.

Lived ListModel

```
public class LiveOrderListModel extends AbstractListModel<Order> {

    private OrderDao orderDao;
    private Integer totalSize;
    private int pageSize = 30;
    private final String CACHE_KEY=
LiveOrderListModel.class+"_cache";
```

```
public LiveOrderListModel(OrderDao orderDao) {
    this.orderDao = orderDao;
}

/**
 * query one page size of entity for one execution a time.
 */
@Override
public Order getElementAt(int index) {
    Map<Integer, Order> cache = getCache();

    Order targetOrder = cache.get(index);
    if (targetOrder == null){
        //if cache doesn't contain target object, query a
page starting from the index
        List<Order> pageResult = orderDao.findAll(index, pageSize);
        int indexKey = index;
        for (Order o : pageResult ){
            cache.put(indexKey, o);
            indexKey++;
        }
    }else{
        return targetOrder;
    }

    //get the target after query from database
    targetOrder = cache.get(index);
    if (targetOrder == null){
        //if we still cannot find the target object from
database, there is inconsistency in the database
        throw new HibernateException("Element at index
"+index+" cannot be found in the database.");
    }else{
        return targetOrder;
    }
}

private Map<Integer, Order> getCache(){
    Execution execution = Executions.getCurrent();
    //we only reuse this cache in one execution to avoid
accessing detached objects.
    //our filter opens a session during a HTTP request
    Map<Integer, Order> cache = (Map)execution.getAttribute(CACHE_KEY);
    if (cache == null){
        cache = new HashMap<Integer, Order>();
        execution.setAttribute(CACHE_KEY, cache);
    }
}
```

```
        }

        return cache;
    }

    @Override
    public int getSize() {
        if (totalSize == null) {
            totalSize = orderDao.findAllSize().intValue();
        }
        return totalSize;
    }
}
```

- Line 16: If the cache doesn't contain target Order, we query a one page size of Order starting from the index as passed `index` doesn't always increase sequentially.
- Line 42: The `getElementAt (int)` will be invoked multiple times during an execution. In order to avoid using a cache of detached objects, we make the cache as an attribute of an execution which is dropped after being handled.

Get Example Source Code

All source code used in this chapter can be found here ^[11].

Reference

- [1] Hibernate in Action, Christian Bauer, Gavin King, Manning
- [2] <http://www.hibernate.org/docs>
- [3] Unit of Work in Hibernate Core Reference Manual (http://docs.jboss.org/hibernate/core/3.6/reference/en-US/html_single/#transactions-basics-uow)
- [4] Open Session in View\Problem (https://community.jboss.org/wiki/OpenSessionInView#The_problem)
- [5] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkplus/hibernate/HibernateUtil.html#>
- [6] Hibernate Reference Documentation\ Tutorial (http://docs.jboss.org/hibernate/core/3.6/reference/en-US/html_single/#tutorial-firstapp Helpers)
- [7] Open Session in View\Using an interceptor (https://community.jboss.org/wiki/OpenSessionInView#Using_an_interceptor)
- [8] Hibernate Reference Documentation\ contextual session (http://docs.jboss.org/hibernate/orm/4.1/manual/en-US/html_single/#architecture-current-session)
- [9] The reason is explained in a Hibernate article "Open Session in View" (https://community.jboss.org/wiki/OpenSessionInView#Why_cant_Hibernate_just_load_objects_on_demand)
- [10] <https://code.google.com/p/zkbooks/source/browse/trunk/developersreference/integration.hibernate/src/main/webapp/homemade/order.zul>
- [11] <https://github.com/zkoss/zkbooks/tree/master/developersreference/integration.hibernate>

JPA

Overview

Java Persistence API (JPA) is a POJO-based persistence specification. It offers *object-relational mapping* solution to enterprise Java applications. We will demonstrate examples on how to integrate a popular and commonly used combination: JPA & Spring. In our example project, we use a popular **JPA 2.0** implementation, **Hibernate 4.0.0.Final**. We will also talk about solutions against well-known `LazyInitializationException`.

Integrate with Spring and JPA

Data Access Object (DAO) pattern is a good practice to implement a persistence layer. This pattern encapsulates persistence API in a DAO object and exposes the DAO object's interface to a business object to perform persistence operations relating to a particular persistent entity.

According to Hibernate EntityManager User Guide's suggestion ^[1], we should apply **entitymanager-per-request** pattern (aka. *Open Session in View* pattern) to manage entity manager. This pattern needs an interceptor to create a new EntityManager when a request is sent to the server and a DAO object would use the same EntityManager to perform persistence operation. We then close the EntityManager before a response is sent to the client. The challenge here is how to implement this pattern. The good news is that we can achieve this through Spring's **OpenEntityManagerInViewFilter** and dependency injection.

Applying this pattern also solves a common **LazyInitializationException** problem most developers may encounter when using a lazy fetching strategy. In brief, EntityManager is usually closed after a DAO object has performed an operation under **entitymanager-per-operation** pattern. Those persistent objects queried by EntityManager become *detached* after associated EntityManagers are closed. If we access a detached object's lazy-loaded collection when rendering the view, we will get an error message like `LazyInitializationException: no session or session was closed`. This problem is essentially identical to a lazy-loading problem in Hibernate. As we apply **entitymanager-per-request** pattern, an EntityManager is kept open when a View accessing a lazy-loaded collection. Those objects queried by EntityManager become detached later (after the interceptor closes the EntityManager) and as a result, the problem mentioned previously is now resolved.

Configuration

The minimal Maven dependencies you need are:

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>3.0.7.RELEASE</version>
    <scope>compile</scope>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>4.0.0.Final</version>
    <scope>compile</scope>
</dependency>
```

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>3.0.7.RELEASE</version>
    <scope>compile</scope>
</dependency>
<dependency>
    <groupId>cglib</groupId>
    <artifactId>cglib</artifactId>
    <version>2.2</version>
    <scope>compile</scope>
</dependency>
```

 **Note:** If you don't use Maven, please refer to JPA vendor's documentation to know which JAR file you need.

Our example project's Spring configuration is for non-managed environment.

Spring configuration

```
<!-- omit headers -->

<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource" >
    <property name="driverClassName" value="org.hsqldb.jdbcDriver" />
    <property name="url" value="jdbc:hsqldb:file:data/store" />
    <property name="username" value="sa" />
    <property name="password" value="" />
</bean>

<bean id="entityManagerFactory" class="org.springframework.orm.jpa.LocalEntityManagerFactoryBean">
    <property name="persistenceUnitName" value="order"/>
</bean>

<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="entityManagerFactory" />
</bean>

<tx:annotation-driven />

<context:component-scan base-package="org.zkoss.reference.developer.jpa" />
```

OpenEntityManagerInViewFilter

To apply *entitymanager-per-request* pattern, we can use Spring provided OpenEntityManagerInViewFilter instead of writing our own one. Make sure filter mapping's url-pattern covers all pages that access lazy-loaded entities. If you don't want this filter intercepting all pages, be sure to include ZK AU request path (/zkau/*) in url-pattern as your event handling methods (or command methods) might also access lazy-loaded objects.

Configure OpenEntityManagerInViewFilter in web.xml

```
<filter>
    <filter-name>OpenEntityManagerInViewFilter</filter-name>

    <filter-class>org.springframework.orm.jpa.support.OpenEntityManagerInViewFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>OpenEntityManagerInViewFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

DAO Implementation

In the DAO, we can easily retrieve EntityManager by Spring's dependency inject without writing any utility class. Spring's declarative transaction management and rollback rule also reduces our work.

DAO empowered by Spring

```
@Repository
public class SpringOrderDao {
    @PersistenceContext
    private EntityManager em;

    @Transactional(readOnly=true)
    public List<Order> queryAll() {
        Query query = em.createQuery("from Order as o");
        List<Order> result = query.getResultList();
        return result;
    }

    @Transactional
    public Order save(Order newOrder) {
        em.persist(newOrder);
        em.flush();
        return newOrder;
    }

    //...
}
```

- Line 4: Spring will inject EntityManager created by **OpenEntityManagerInViewFilter**.

Lazy Initialization Issue among AU Requests

We apply *open session in view* pattern to keep an EntityManager open after a page is rendered, this makes a ZUL that accesses a lazy-loaded collection to be rendered normally **when you visit the ZUL at first request**. However, if your event handling methods (or command methods) access lazy-loaded collection of another detached object, you still get LazyInitializationException when a user interacts with the ZUL. This is because even though the filter opens an EntityManager for each request, the detached objects don't attach to the EntityManager automatically.^[2] There are two solutions for this problem.

1. set fetching strategy to **eagerly fetch**.
2. **Re-query detached object** manually.

If you don't have large amount of data, you can choose the first solution; just change your fetching strategy to **eager** for one-to-many mapping.

We will demonstrate the **second solution** here using an example assuming that we have a "Order Viewer" system where we can view an order and its details. The upper *Listbox* contains a list of orders and the lower *Grid* contains the details the selected order. One order may contain many order items (one-to-many mapping), and we set order items collection to lazy fetching . When we select an order, the *Grid* displays details of the selected order which means accessing a lazy-loaded collection.

Order and OrderItem

```
@Entity
@Table(name="orders")
public class Order {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String status = PROCESSING;
    private String description;

    @OneToMany(mappedBy="orderId", fetch=FetchType.LAZY)
    private List<OrderItem> items = new ArrayList<OrderItem>();

    //other codes...
}
```

We set *Listbox* to select the first row of the orders as default. When the ZUL accesses the selected order's lazy-loaded items collection, JPA can load it successfully with the help of OpenEntityManagerInViewFilter because EntityManager is still open. However, if we click the second row which accesses a detached Order object's items collection, we should re-load the Order object with JPA EntityManager or we'll get LazyInitializationException.

The screenshot shows a user interface titled "Order Viewer". At the top, there is a table with three columns: "ID", "Description", and "Status". Two rows are visible: one for "nobody at home in day time" (status: processing) and another for "urgent case" (status: processing). A cursor points to the second row. Below this, a section titled "Details" contains a table with three columns: "Name", "Quantity", and "Price". It lists two items: "Cookies" (quantity: 10, price: 4.0) and "Toast" (quantity: 5, price: 3.1).

source of above screen [3]

In order to reload a detached object, we pass the selected order to DAO object and reload it.

Reload selected object

```
public class SpringOrderViewModel {
    @WireVariable
    private SpringOrderDao springOrderDao;

    private List<Order> orders ;
    private Order selectedItem;

    @Init
    public void init(){
        orders = springOrderDao.queryAll();
        if (!orders.isEmpty()){
            setSelectedItem(orders.get(0));
        }
    }

    public Order getSelectedItem() {
        if (selectedItem!=null){
            selectedItem = springOrderDao.reload(selectedItem);
            //you could replace the item in model list with
            persistent one
        }
        return selectedItem;
    }

    //omit other codes
}
```

- Line 12: Initialize the *Listbox* selection with the `Order` object at index 0 of `orders`.
- Line 17: Re-query the `selectedItem`.

We reload the detached `Order` objects from the database, this will make the detached object attach to an `EntityManager`. Then, when we access the lazy-loaded collection (`items`), JPA can retrieve the collection for us. After doing so, we can eliminate `LazyInitializationException`.

Reload detached object

```
@Repository
public class SpringOrderDao {

    @PersistenceContext
    private EntityManager em;

    //omit other codes

    @Transactional(readOnly=true)
    public Order reload(Order order) {
        return em.find(Order.class, order.getId());
    }

}
```

Lazy Initialization Issue Under Render on Demand

Some AU requests cannot be interfered by developers like "Render On Demand" request. The rendering request is handled implicitly by a component itself. Under this situation, if a component needs to **render some data from a detached object's lazy-loaded collection**, developers won't have a chance to reload detached objects during rendering to avoid `LazyInitializationException`. Let's use an example to explain this situation.

Assume we have a *Listbox*, it only displays 10 rows by default and it's not in "page" mold. One of its columns display a lazy-loaded collection's size (`each.items.size()`) of an `Order` object.

Listbox that accesses lazy-loaded property

```
<window title="" border="normal" width="600px" apply="org.zkoss.bind.BindComposer"
viewModel="@id('vm')"

@init('org.zkoss.reference.developer.jpa.vm.RodViewModel')>
    Contain a customized model that reload lazy-loaded
collection from database
    <listbox model="@load(vm.orderListModel)" rows="10">
        <listhead>
            <listheader label="ID" width="50px" />
            <listheader label="Description" />
            <listheader label="Item Count" width="150px" />
        </listhead>
        <template name="model">
            <listitem>
                <listcell label="@load(each.id) " />
                <listcell label="@load(each.description) " />
```

```

        <listcell label="@load(each.items.size())" />
    </listitem>
</template>
</listbox>
...

```

If we just pass a Java `List` object to be the model of the `Listbox`, when a user scrolls down to view other rows, ZK will send AU request to retrieve data for those un-rendered rows. `Listbox` will try to access lazy-loaded collection but objects in the list are already detached, and we will get `LazyInitializationException`. During this rendering process, developers will not be notified and cannot interfere this process to reload detached objects. One solution is to **implement a custom ListModel**^[7] for the component.

We demonstrate 2 implementations here for your reference. The first one is simpler but less efficient; it re-queries each detached object when a component requests it.

Reloaded ListModel

```

public class OrderListModel extends AbstractListModel<Order> {

    private SpringOrderDao orderDao;
    List<Order> orderList = new LinkedList<Order>();

    public OrderListModel(List<Order> orders, OrderDao orderDao) {
        this.orderList = orders;
        this.orderDao = orderDao;
    }

    @Override
    public Order getElementAt(int index) {
        //throw a runtime exception if orderDao does not find
target object
        Order renewOrder = orderDao.reload(orderList.get(index));
        return renewOrder;
    }

    @Override
    public int getSize() {
        return orderList.size();
    }
}

```

- Line 1: We extend `AbstractListModel`^[6] to build our list model for it to handle selection for us, but we have to override `Order`'s `equals()` and `hashCode()`.
- Line 14: Re-query the detached object by its id and return a persistent one.

The second one is more complicated but more efficient; it re-queries a one page size data each time and stores as a cache in an execution. If the cache has the object that the component requests, it returns the one in cache without re-querying it again.

Lived ListModel

```
public class LiveOrderListModel extends AbstractListModel<Order>{

    private SpringOrderDao orderDao;
    private Integer totalSize;
    private int pageSize = 30;
    private static final String CACHE_KEY=
LiveOrderListModel.class+"_cache";

    public LiveOrderListModel(OrderDao orderDao) {
        this.orderDao = orderDao;
    }

    /**
     * query one page size of entity for one execution a time.
     */
    @Override
    public Order getElementAt(int index) {
        Map<Integer, Order> cache = getCache();

        Order targetOrder = cache.get(index);
        if (targetOrder == null){
            //if cache doesn't contain target object, query a
page starting from the index
                List<Order> pageResult = orderDao.findAll(index, pageSize);
                int indexKey = index;
                for (Order o : pageResult ){
                    cache.put(indexKey, o);
                    indexKey++;
                }
        }else{
            return targetOrder;
        }

        //get the target after query from database
        targetOrder = cache.get(index);
        if (targetOrder == null){
            //if we still cannot find the target object from
database, there is inconsistency in the database
                throw new RuntimeException("Element at index
"+index+" cannot be found in the database.");
        }else{
            return targetOrder;
        }
    }

    private Map<Integer, Order> getCache(){
        Execution execution = Executions.getCurrent();
```

```
//we only reuse this cache in one execution to avoid
accessing detached objects.

//our filter opens a session during a HTTP request
Map<Integer, Order> cache = (Map)execution.getAttribute(CACHE_KEY);
if (cache == null) {
    cache = new HashMap<Integer, Order>();
    execution.setAttribute(CACHE_KEY, cache);
}
return cache;
}

@Override
public int getSize() {
    if (totalSize == null) {
        totalSize = orderDao.queryAllSize().intValue();
    }
    return totalSize;
}
}
```

- Line 16: If the cache doesn't contain target Order, we query one page size of Order starting from the index because passed index doesn't always increase sequentially.
- Line 42: The getElementAt (int) will be invoked multiple times during an execution. In order to avoid using a cache of detached objects, we make the cache as an attribute of an execution which is dropped after being handled.

Get Example Source Code

All source code used in this chapter can be found here ^[4].

Reference

- [1] Unit of Work in Hibernate EntityManager User Guide (http://docs.jboss.org/hibernate/entitymanager/3.6/reference/en/html_single/#transactions-basics-uow)
- [2] The reason is explained in a Hibernate article "Open Session in View" (https://community.jboss.org/wiki/OpenSessionInView#Why_cant_Hibernate_just_load_objects_on_demand)
- [3] <https://code.google.com/p/zkbooks/source/browse/trunk/developersreference/integration.jpa/src/main/webapp/order.zul>
- [4] <https://github.com/zkoss/zkbooks/tree/master/developersreference/integration.jpa>

Security

In the following sections, we are going to describe how to integrate security frameworks with ZK.

Spring Security

Overview

Spring Security is an application framework that provides security services for J2EE-based enterprise software application. It is a popular and widely adopted framework, in this article we will demonstrate how to integrate it to secure a ZK application including securing pages, handling authentication process, securing components, and securing events. Our example is a simple forum-like application. Users can read, create, edit, and delete an article according to his authorities.

Configuration

Maven

We need to add dependencies for Spring Security and Maven's transitive dependency management can include all necessary dependencies of Spring for us.

```
<!-- Spring Security -->
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
    <version>${springsecurity.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-web</artifactId>
    <version>${springsecurity.version}</version>
</dependency>

<!-- extra -->
<dependency>
    <groupId>commons-logging</groupId>
    <artifactId>commons-logging</artifactId>
    <version>1.1.1</version>
</dependency>
<dependency>
    <groupId>cglib</groupId>
    <artifactId>cglib</artifactId>
    <version>2.2</version>
</dependency>
```

- Line 4: Because we use the security namespace in the application context, we need `spring-security-config`.
- Line 16: Spring-core depends on commons-logging.
- Line 21: The cglib is optional. We add it because we use CGLIB-based class proxy.

Note: If you don't use Maven, please refer to Spring Security Reference Documentation to check which JAR files are needed.

Spring

Our example application also integrates Spring framework, the required configuration in `web.xml` is as follows:

web.xml

```
<!-- Loads the Spring application context configuration -->
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<!-- For using web scoped bean -->
<listener>
    <listener-class>org.springframework.web.context.request.RequestContextListener</listener-class>
</listener>
```

The `ContextLoaderListener` will load `/WEB-INF/applicationContext.xml` (Spring configuration file) by default, and we follow this convention so we don't need to add extra configuration in `web.xml`.

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation=
           "http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
            http://www.springframework.org/schema/aop
            http://www.springframework.org/schema/aop
            http://www.springframework.org/schema/aop/spring-aop-2.0.xsd">

    <context:component-scan base-package="org.zkoss.reference.developer.spring.security.model"/>

    <import resource="applicationContext-security.xml"/>
</beans>
```

- Line 9: We can register beans by class-path scanning to reduce XML configuration effort.
- Line 11: We can import another configuration file for Spring Security.

Security Namespace Configuration

The first configuration you should add to use Spring Security is a filter declaration in web.xml:

```
<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

This filter is a hook into Spring Security's web infrastructure. It intercepts all requests and hands over them to be processed by Spring Security internal filters.

Namespace configuration has been supported by Spring framework since version 2.0 and it is an alternative configuration syntax which is closer to problem domain. It also can reduce configuration's complexity because one element may contain multiple beans and processing steps. To use the security namespace, you should have `spring-security-config` in your classpath and add the schema declaration to your application context file:

applicationContext-security.xml

```
<beans:beans xmlns="http://www.springframework.org/schema/security"
    xmlns:beans="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation=
        "http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.0.xsd
        http://www.springframework.org/schema/security
        http://www.springframework.org/schema/security/spring-security-3.1.xsd">

    <!-- HTTP configuration sample -->
    <http auto-config="true">
        <!-- ZK AU request -->
        <intercept-url pattern="/zkau/**" access="IS_AUTHENTICATED_ANONYMOUSLY" />
        <!-- the login page -->
        <intercept-url pattern="/login.zul" access="IS_AUTHENTICATED_ANONYMOUSLY" />
        <!-- pages for anonymous access in an application -->
        <intercept-url pattern="/index.zul" access="IS_AUTHENTICATED_ANONYMOUSLY" />
        <intercept-url pattern="/articleContent.zul" access="IS_AUTHENTICATED_ANONYMOUSLY" />
        <!-- secure pages -->
        <intercept-url pattern="/**" access="ROLE_USER" />

        <form-login login-page="/login.zul"
            authentication-failure-url="/login.zul?login_error=1"
```

```

        login-processing-url="/j_spring_security_check"/>

    <logout logout-success-url="/index.zul" invalidate-session="true" />
</http>

<!-- omit inactive configurations -->

<authentication-manager>
    <authentication-provider user-service-ref="myUserDetailsService">
        <password-encoder hash="md5" />
    </authentication-provider>
</authentication-manager>

</beans:beans>

```

Here we introduce some main elements and will leave the details in the subsequent sections.

- Line 12: The `<http>` element is the parent for all web-related namespace functions and we use `auto-config` to save configuration efforts. We also create a HTTPS configuration sample in `applicationContext-security.xml`. Please see source code for details.
- Line 32: Each Spring Security application which uses the namespace configuration must include `<authentication-manager>`. It is responsible for registering the `AuthenticationManager` which provides authentication services to the application.
- Line 33: We implement our `MyUserDetailsService` bean to provide authentication service and configure it in `<authentication-provider>` element.

Secure Pages

In Spring Security, pages are protected by `<intercept-url>` element under `<http>`. We can specify in `pattern` to match against the URLs of incoming requests using an ant path style syntax in `<intercept-url>` element. The `access` attribute defines the access permission for requests which match the given pattern. Here we use simple role-based access control.

In most cases, we usually secure all pages with :

```
<intercept-url pattern="/**" access="ROLE_USER" />
```

The "ROLE_USER" is an authority string we define and give for each authenticated user in our custom user service, `MyUserDetailsService`.

Then we can selectively allow some pages for anonymous access like:

```
<intercept-url pattern="/login.zul" access="IS_AUTHENTICATED_ANONYMOUSLY" />
```

`IS_AUTHENTICATED_ANONYMOUSLY` is a built-in permission value used to grant access to anonymous users

applicationContext-security.xml

```

<!-- HTTP configuration sample -->
<http auto-config="true">
    <!-- ZK AU requests -->
    <intercept-url pattern="/zkau/**" access="IS_AUTHENTICATED_ANONYMOUSLY" />
    <!-- the login page -->
    <intercept-url pattern="/login.zul" access="IS_AUTHENTICATED_ANONYMOUSLY" />
    <!-- pages for anonymous access in an application -->

```

```

<intercept-url pattern="/index.zul" access="IS_AUTHENTICATED_ANONYMOUSLY" />
<intercept-url pattern="/articleContent.zul" access="IS_AUTHENTICATED_ANONYMOUSLY" />
<!-- secure pages -->
<intercept-url pattern="/**" access="ROLE_USER" />

...
</http>

```

- Line 4: ZK AU requests must be available for anonymous access or ZK can't work normally.
- Line 6: Remember to set login page URL available to anonymous users, otherwise users won't be able to access the log-in page (this is a common configuration error).
- Line 14: Restrict all page requests with permission ROLE_USER.

Authentication

Setting `auto-config` enables form-based login process automatically but it uses Spring Security's built-in login page. We usually build our own login page so we can specify our custom login page URL to override the default configuration.

`applicationContext-security.xml`

```

<!-- HTTP configuration sample -->
<http auto-config="true">
    ...
    <form-login login-page="/login.zul"
                authentication-failure-url="/login.zul?login_error=1"
            />
    <logout logout-success-url="/index.zul" invalidate-session="true" />
</http>

```

- Line 5: Specify the URL used to render the login page at `login-page`.
- Line 5: Specify the URL to redirect the browser on login failure,
- Line 8: Specify the destination URL in which the user will be redirected to after logging out.

In order to let Spring Security handle authentication, we should use HTML's form in a zul.

`login.zul`

```

...
<html:form action="j_spring_security_check" method="POST"
           xmlns:html="native">
    <grid>
        <rows>
            <row>User: <textbox id="u" name="j_username"/></row>
            <row>Password: <textbox id="p" type="password" name="j_password"/></row>
            <row spans="2">
                <hbox>
                    <button type="reset" label="Reset" />
                    <button type="submit" label="Submit" />
                </hbox>
            </row>
        </rows>
    </grid>
</html:form>

```

```

        </row>
    </rows>
</grid>
</html:form>

```

- Line 2: The default action URL monitored by Spring Security filter is `j_spring_security_check`.
- Line 6,7: The login form should contain `j_username` and `j_password` input fields.

In most cases, each application will have its own way to authenticate a user and Spring Security provides various authentication provider to achieve it. We can create a simple `MyUserDetailsService` which implements Spring Security's `UserDetailsService` interface to perform our own authentication.

MyUserDetailsService

```

@Service
public class MyUserDetailsService implements UserDetailsService {

    private static final Map<String, MyUser> USERS = new HashMap<String, MyUser>();
    private static void add(MyUser mu) {
        USERS.put(mu.getUsername(), mu);
    }
    static{

        add(new MyUser("rod", "81dc9bdb52d04dc20036dbd8313ed055",
//password:1234
                new String[]{ "ROLE_USER", "ROLE_EDITOR" } ));

        add(new MyUser("dianne", "81dc9bdb52d04dc20036dbd8313ed055",
                new String[]{ "ROLE_USER", "ROLE_EDITOR" } ));

        add(new MyUser("scott", "81dc9bdb52d04dc20036dbd8313ed055",
                new String[]{ "ROLE_USER" } ));

        add(new MyUser("peter", "81dc9bdb52d04dc20036dbd8313ed055",
                new String[]{ "ROLE_USER" } ));
    }

    // must return a value or throw UsernameNotFoundException
    public UserDetails loadUserByUsername(String username)
            throws UsernameNotFoundException {
        //perform authentication
    }
}

```

- Line 10: `MyUser` extends `org.springframework.security.core.userdetails.User` which implements `org.springframework.security.core.userdetails.UserDetails`.
- Line 11: Here we define two authorities, `ROLE_USER` and `ROLE_EDITOR`.

Then specify `user-service-ref` with our `MyUserDetailsService`.

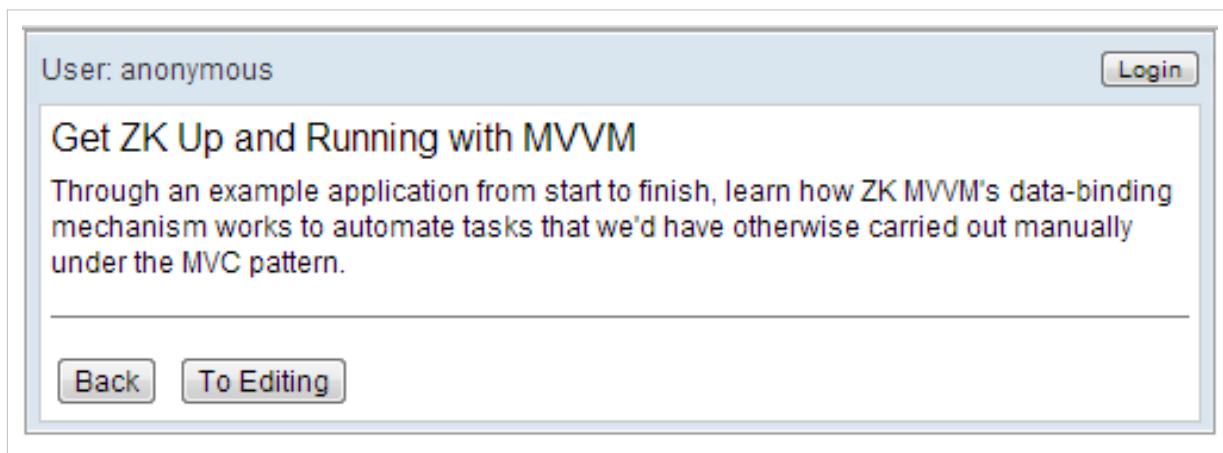
applicationContext-security.xml

```
<authentication-manager>
    <authentication-provider user-service-ref="myUserDetailsService">
        <password-encoder hash="md5" />
    </authentication-provider>
</authentication-manager>
```

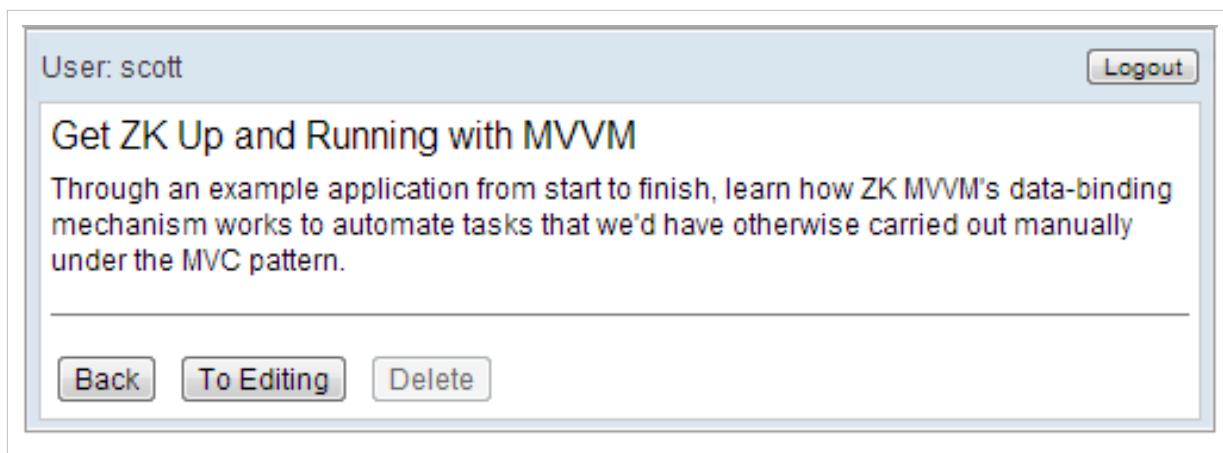
- Line 2: Configure Spring Security to use our custom user service.

Secure Components

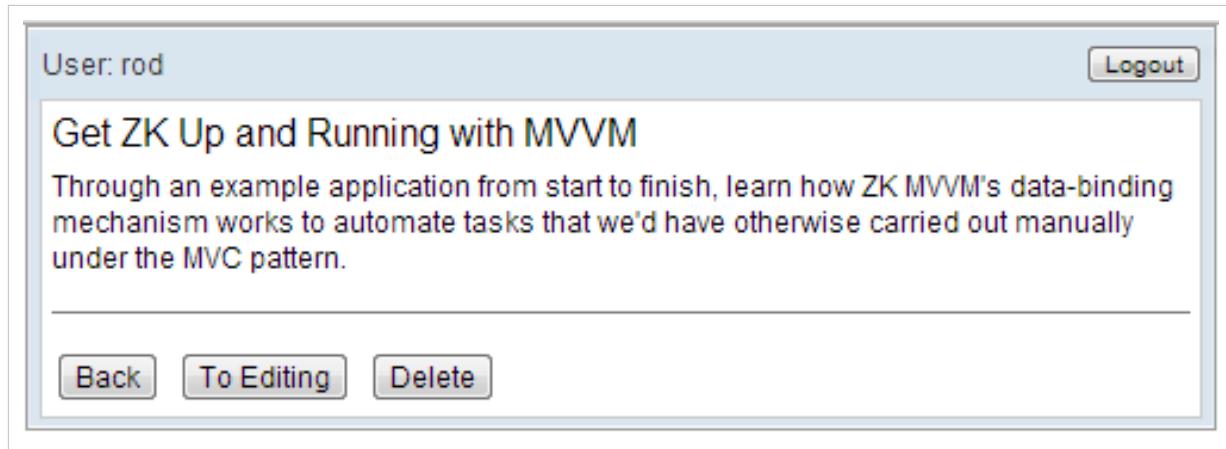
Every authenticated user has his/her own authorities. A common scenario is where we want to control UI's status according to current user's authorities. In our example, an anonymous user can only view an article, and a user with "ROLE_USER" can see a disabled "Delete" button. But a user with "ROLE_EDITOR" can see the "Delete" button and be able to click it.



What a anonymous user see



What a user with "ROLE_USER" see



What a user with "ROLE_EDITOR" see

How do we achieve this security control in a zul? We can implement a custom tag library to check current user's authorities and apply the tag library on if or disable attribute. Please refer to ZUML_Reference/ZUML/Processing_Instructions/taglib/Custom_Taglib for details.

The tag library's function is implemented in `org.zkoss.reference.developer.spring.security.SecurityUtil`. Here we just briefly explain how `isAllGranted()` is implemented and you can read other methods in the source code .

The `SecurityContextHolder` is the most fundamental object and is where Spring Security stores the present security context of the application, which includes details of the principal currently using the application. We can obtain user information from `SecurityContextHolder` including his authorities.

SecurityUtil.java

```
package org.zkoss.reference.developer.spring.security;

//omit import

public class SecurityUtil {

    public static boolean isAllGranted(String authorities) {
        if (null == authorities || "".equals(authorities)) {
            return false;
        }
        final Collection<? extends GrantedAuthority> granted = getPrincipalAuthorities();
        boolean isAllGranted =
granted.containsAll(parseAuthorities(authorities));
        return isAllGranted;
    }

    private static Collection<? extends GrantedAuthority> getPrincipalAuthorities() {
        Authentication currentUser =
SecurityContextHolder.getContext().getAuthentication();
        if (null == currentUser) {
            return Collections.emptyList();
        }
        if ((null == currentUser.getAuthorities()) ||
            (null == currentUser.getAuthorities().size()))
            return Collections.emptyList();
        else
            return currentUser.getAuthorities();
    }
}
```

```

        (currentUser.getAuthorities().isEmpty()) {
            return Collections.emptyList();
        }
        Collection<? extends GrantedAuthority> granted = currentUser.getAuthorities();
        return granted;
    }

    //omit other methods
}

```

- Line 7: Return true if the authenticated principal is granted ALL of the roles specified in authorities. The input parameter is a comma separated list of roles which the user have been granted.

Then we still have to write a description file to describe the functions that we can use in a zul. In our example, it is /WEB-INF/security.tld. You can read the file in the source code.

security.tld

```

<taglib>
    <uri>http://www.zkoss.org/demo/integration/security</uri>
    <description>
        Methods and actions for ZK + Spring Security
    </description>

    <function>
        <name>isAllGranted</name>
        <function-class>org.zkoss.reference.developer.spring.security.SecurityUtil</function-class>
        <function-signature> boolean isAllGranted(java.lang.String authorities)
        </function-signature>
        <description>
            Return true if the authenticated principal is granted
            authorities of ALL the specified roles.
        </description>
    </function>
    ...
</taglib>

```

Before using a tag library in a zul, we should load its tld file with a directive.

articleContent.zul

```

<?taglib uri="http://www.zkoss.org/zkspring/security" prefix="sec"?>
...
<button id="deleteBtn" label="Delete"
        if="${sec:isAllGranted('ROLE_USER')}"
        disabled="${not sec:isAllGranted('ROLE_EDITOR')}"/>

```

- Line 1: Use a directive to load a custom tag library's tld file.
- Line 5: Hide this button for anonymous users.
- Line 6: Disable the button for those users without ROLE_EDITOR authority.

The function `isAllGranted()` will return true if the authenticated principal is granted all of the roles specified in authorities. An anonymous user doesn't have authority "ROLE_USER", so the "Delete" button will not be created. If a user have authority "ROLE_EDITOR", he can see an enabled "Delete" button (the `disabled` will be `false`). You can use our `SecurityUtil` and `security.tld` as reference and write your own one to apply in your application.

Secure Events

If you want to restrict available actions according to a business rule or a dynamic status, this cannot be achieved by tag library. To do this, Spring Security provides a "method security" which can add security to your service layer methods. To use this feature, you should declare as follows:

applicationContext-security.xml

```
<global-method-security secured-annotations="enabled" />
```

Then add `@Secure` to those methods you want to secure with permissions.

ArticleService.java

```
public interface ArticleService {  
  
    @Secured({"ROLE_USER", "IS_AUTHENTICATED_ANONYMOUSLY"})  
    public List<Article> findAll();  
  
    @Secured({"ROLE_USER", "IS_AUTHENTICATED_ANONYMOUSLY"})  
    public Article find(long id);  
  
    @Secured({"ROLE_USER"})  
    public void create(Article a);  
  
    @Secured({"ROLE_EDITOR", "ROLE_USER"})  
    public void update(Article a);  
  
    @Secured({"ROLE_EDITOR"})  
    public void delete(long id);  
}
```

If a user uses the service and has no permission, Spring Security will throw its `AccessDeniedException`.

If the security checking is more dynamic and cannot be determined in compile time, we can check a user's permission in an event listener of a controller. In our example, a user with "ROLE_EDITOR" can edit any article but a user with "ROLE_USER" can only edit those articles written by himself/herself. We can check this when a user clicks the "Edit" button:

permission checking in an event listener

```
@VariableResolver(DelegatingVariableResolver.class)  
public class ArticleContentViewController extends SelectorComposer<Component> {  
  
    //omit other methods  
  
    @Listen("onClick=#openEditorBtn")
```

```

public void edit() {
    //ownership & permission check.
    if(!isOwner() ||
SecurityUtil.isAllGranted("ROLE_EDITOR")) {
        throw new AccessDeniedException(
            "The user is neither the author, nor a
privileged user.");
    }
    ArticleEditor editor = new ArticleEditor();
    editor.setParent(container);
    editor.doHighlighted();
}
}

```

- Line 9~11: If the current login user is neither the owner of the article nor has the authority "ROLE_EDITOR", we will not allow the editing of the article and throw a Spring Security's AccessDeniedException.

If we throw a runtime exception for an access with insufficient permission, ZK will show the error message on the page by default. But for an unauthenticated user (not log in yet), we can even do more: redirect the anonymous user to the login page. We will show how to achieve this in ZK:

First, we have to catch the exception thrown in an event listener by ZK error handling mechanism, configure `<error-page>` in `zk.xml`.

zk.xml

```

<error-page>
    <exception-type>org.springframework.security.access.AccessDeniedException</exception-type>
    <location>/WEB-INF/errors/handleAccessDenied.zul</location>
</error-page>

```

Then, create the error handling page. To avoid users visiting the page directly, we put it under /WEB-INF. This error handling page displays nothing but a page initiator to redirect an unauthenticated user to the login page.

handleAccessDenied.zul

```

<?init class="org.zkoss.reference.developer.spring.security.ui.error.AjaxAccessDeniedHandler"?>
<zks>
<!-- Forward a unauthenticated user to login page. -->
</zks>

```

The page initiator redirects the browser to login page if current user principle doesn't exist, otherwise, it displays exception's detail in a custom page.

AjaxAccessDeniedHandler.java

```

public class AjaxAccessDeniedHandler extends GenericInitiator {

    public void doInit(Page page, Map<String, Object> args) throws Exception {
        // when this initiator has been executed that means users
encounter access denied problem.

        Execution exec = Executions.getCurrent();
    }
}

```

```
if (null == SecurityUtil.getUser()) { //unauthenticated user
    exec.sendRedirect("/login.zul");
} else {
    //display error's detail

Executions.createComponents("/WEB-INF/errors/displayAccessDeniedException.zul",
    null, args);
}
}
```

Login Page

If you implement a login page with zul, you need to configure Spring Security filter to ignore all AU requests (starting with /zkau). Because ZK framework communicates with a server via that channel for various things including firing events and getting resources. If the security filter blocks AU requests, your login page cannot work.

But not intercepting ZK AU requests is equivalent to opening an unchecked channel, which may result in security vulnerabilities.

Non-ZK login entrypoint

Hence, to ensure all zk components are under the security filter's protection, a good practice is to implement authentication without relying on ZK.

This can be done with form-based authentication by implementing the login form in HTML or JSP. And this can also be achieved with other authentication methods which don't require a ZK page to be served, such as redirecting to an external provider, using a SSO provider, etc.

With these options, the security filter doesn't have to be configured to allow anonymous access to ZK AU requests. After a user logs in, all requests including ZK AU can be intercepted by the security filter and rejected if the user is unauthorized or unauthenticated.

Example Source Code

All source code of examples used in this chapter can be found in here ^[1].

References

[1] <https://github.com/zkoss/zkbooks/tree/master/developersreference/integration.spring.security>

Miscellaneous

In following sections, we will talk about other integration issues which are not covered and does not classify in previous sections.

Google Analytics

To track the Ajax traffic with Google Analytics [1] or other statistic services, you have to override a client-side API: zk.Event, zk.Desktop) zAu.beforeSend(_global_.String, zk.Event, zk.Desktop) [2]. This method will be called each time ZK Client is about to send an Ajax request to the server. You could override it to record the requests on any statistic service you prefer.

Here we use Google Analytics as an example to illustrate how to override it.

```
try {
var pageTracker = _gat._getTracker("UA-xxxx");
//whatever code your
website is assigned
pageTracker._setDomainName("zkoss.org");
pageTracker._initData();
pageTracker._trackPageview();

zk.override(zAu, "beforeSend", function (uri, req) {
    try {
        var t = req.target;
        if (t && t.id && (!req.opts || !req.opts.ignorable)) {
            var data = req.data||{},
                value = data.items &&
data.items[0]?data.items[0].id:data.value;
            pageTracker._trackPageview(uri + "/" + t.id + "/" +
req.name + (value?"/"+value:""));
        }
    } catch (e) {
    }
    return zAu.$beforeSend(uri, req);
});
} catch (e) {
}
```

Of course, you could only record the information you are interested by examining Event.name [3].

References

- [1] <http://www.google.com/analytics/>
- [2] [http://www.zkoss.org/javadoc/latest/jsdoc/_global/_zAu.html#beforeSend\(_global_.String](http://www.zkoss.org/javadoc/latest/jsdoc/_global/_zAu.html#beforeSend(_global_.String)
- [3] <http://www.zkoss.org/javadoc/latest/jsdoc/zk/Event.html#name>

Start Execution in Foreign Ajax Channel

Employment/Purpose

Here we describe how to start a ZK execution in a foreign Ajax channel. For example, JSF 2 allows developers to send back JavaScript code to update the browser in JSF's Ajax channel.

Bridge^[1]

Starting an execution in a foreign Ajax channel is straightforward: invoke `javax.servlet.http.HttpServletRequest`, `javax.servlet.http.HttpServletResponse`, `org.zkoss.zk.ui.Desktop`) `Bridge.start(javax.servlet.ServletContext, javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse, org.zkoss.zk.ui.Desktop)`^[2]. Then, you are allowed to access the components, post events and do anything you like. At the end, you invoke `Bridge.getResult()`^[3] to retrieve the JavaScript code snippet and send it back to the client to execute. Finally, you invoke `Bridge.close()`^[4] to close the execution.

```
Bridge bridge = Bridge.start(svlctx, request, response, desktop);
try {
    //execution started, do whatever you want

    String jscode = bridge.getResult();

    //send jscode back with the foreign Ajax channel.
} finally {
    bridge.close(); //end of execution and cleanup
}
```

Example

Start Execution in JSF 2 ActionListener

In JSF 2.0 developers can initiate Ajax request using `jsf.ajax.request`^[5] For e.g.

```
...
<h:commandButton id="save" value="Save"
    onclick="jsf.ajax.request(this, event, {execute:'@all'}); return
false;" actionListener="#{myBean.saveDetails}">
</h:commandButton>
...
```

and in your ActionListener

```
@ManagedBean
@SessionScoped
```

```
public class MyBean {

    public void saveDetails(ActionEvent e) throws IOException {

        ExternalContext ec =
FacesContext.getCurrentInstance().getExternalContext();
        ServletContext svlctx = (ServletContext) ec.getContext();
        HttpServletRequest request = (HttpServletRequest)
ec.getRequest();
        HttpServletResponse response = (HttpServletResponse)
ec.getResponse();
        Component comp = getComponent();
        Bridge bridge = Bridge.start(svlctx, request,
response, comp.getDesktop());
        try {
            // update ZK component(s) state here
            //comp.appendChild(new SomethingElse()); ...

            //Send back bridge.getResult() with the response writer
(eval)
        PartialResponseWriter responseWriter =
FacesContext.getCurrentInstance().getPartialViewContext().getPartialResponseWriter();
            responseWriter.startDocument();
            responseWriter.startEval();
            responseWriter.write(bridge.getResult());
            responseWriter.endEval();
            responseWriter.endDocument();
            responseWriter.flush();
            responseWriter.close();
        } finally {
            bridge.close();
        }
    }

    private Component getComponent() {
        //locate the component that you want to handle
    }
}
```

-
- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkplus/embed/Bridge.html#>
[2] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkplus/embed/Bridge.html#start\(javax.servlet.ServletContext,](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkplus/embed/Bridge.html#start(javax.servlet.ServletContext)
[3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkplus/embed/Bridge.html#getResult\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkplus/embed/Bridge.html#getResult)
[4] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkplus/embed/Bridge.html#close\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkplus/embed/Bridge.html#close)
[5] For more information on jsf.ajax.request (<https://javaserverfaces.dev.java.net/nonav/docs/2.0/jsdocs/symbols/jsf.ajax.html#request>) read official JSF Javascript docs for jsf.ajax (<https://javaserverfaces.dev.java.net/nonav/docs/2.0/jsdocs/symbols/jsf.ajax.html>).

Version History

Version	Date	Content
5.0.5	September 2010	Bridge (http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkplus/embed/Bridge.html#) was introduced to simplify the starting of an execution in foreign Ajax channel

Websocket Channel

Employment/Purpose

ZK has supported a way to share the application data between a ZK application and a websocket application within the same session. Here we demonstrate how to use the Storage ^[1] in a desktop scope to share the application data through the websocket channel.

Example

websocket Server

```
@ServerEndpoint(value ="/echo/",
    configurator = ZKWebSocket.class)
public class EchoServer {
    @OnOpen
    public void onOpen(Session session, EndpointConfig config) {
        //since zk 8.6.4
        ZKWebSocket.initZkDesktop(session, config);
    }

    @OnMessage
    public void onMessage(String message, Session session) {
        Storage<Integer> storage = ZKWebSocket.getDesktopStorage(session);
        if ("receive".equals(message)) {
            Integer count = storage.getItem("count");
            try {
                session.getBasicRemote().sendText("Received..." +
+ count);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

```

        } else {
            try {
                storage.setItem("count",
Integer.parseInt(message));
                session.getBasicRemote().sendText("Sent..." +
message);
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }
    }

    @OnClose
    public void onClose(Session session) {
    }

}

```

As you can see above, in line 2, we have to register a `ZKWebSocket` [2] class into the configurator of the `ServerEndpoint` annotation. And in line 12 we can use the method of `ZKWebSocket.getDesktopStorage(javax.websocket.Session)` [3] to receive the data storage from a websocket session (the storage is a thread-safe implementation). Note that the websocket session must have a `dtid` value which is sent from client as follows.

```
// Create a new instance of the websocket
var webSocket = new
WebSocket("ws://localhost:8080/zkwebsocket/echo/?dtid=" +
zk.$('win').desktop.uuid);
```

ZK Application

MVVM Example

```

<window id="win" apply="org.zkoss.bind.BindComposer"
        viewModel="@id('vm')"
@init('org.zkoss.foo.ZKWebSocketViewModel')>
    <groupbox title="ZK">
        <hlayout>count: <label value="@load(vm.count)" /></hlayout>
        <button label="add" onClick="@command('cmd')"/>
    </groupbox>
</window>

@ToServerCommand("update")
public class ZKWebSocketViewModel {

    private Integer count;

    @Init
    public void init(@ContextParam(ContextType.DESKTOP) Desktop
desktop) {

```

```

        count = 100;
        syncToStorage(desktop);
    }

    @Command
    @NotifyChange("count")
    public void cmd(@ContextParam(ContextType.DESKTOP) Desktop
desktop) {
        count++;
        syncToStorage(desktop);
    }

    @Command("update")
    @NotifyChange("count")
    public void doUpdate(@ContextParam(ContextType.DESKTOP) Desktop
desktop) {
        count = desktop.<Integer>getStorage().getItem("count");
    }

    private void syncToStorage(Desktop desktop) {
        Storage<Integer> desktopStorage = desktop.getStorage();
        desktopStorage.setItem("count", count);
    }

    public Integer getCount() {
        return count;
    }
}

```

As you can see above, in line 22 and 26, we can receive the data storage from the desktop object to share or update the application data into it, so that the websocket echo server can use or get the latest data from it or vice versa.

MVC Example

```

<window id="win" apply="org.zkoss.foo.ZKWebSocketComposer">
    <groupbox title="ZK">
        <hlayout>count: <label id="label" /></hlayout>
        <button id="btn" label="add"/>
    </groupbox>
</window>

public class ZKWebSocketComposer extends SelectorComposer<Window> {
    @Wire Label label;
    @Wire Button btn;
    private Integer count;

    @Override public void doAfterCompose(Window comp) throws
Exception {
    super.doAfterCompose(comp);
    count = 100;
}

```

```

        label.setValue("100");
        syncToStorage();
    }

    @Listen("onClick = #btn")
    public void doClick() {
        count++;
        label.setValue(String.valueOf(count));
        syncToStorage();
    }

    private void syncToStorage() {
        getSelf().getDesktop().getStorage().setItem("count",
count);
    }

    @Command // this annotation is under the package of
org.zkoss.zk.ui.annotation
    public void update() {
        count =
getSelf().getDesktop().<Integer>getStorage().getItem("count");
        label.setValue(String.valueOf(count));
    }
}

```

As you can see above, in line 21 and 26, we can receive the data storage from the desktop object to share or update the application data into it, so that the websocket echo server can use or get the latest data from it or vice versa.

Note: in line 24 Command [4] annotation has been added since the release of ZK 8.0.0, and it is used to receive a notification from client to server. For more details, please take a look at the #Communication section.

Communication

From Websocket server to ZK application

MVVM Example

Here is the MVVM way to send a command from client to server.

```

// Create a new instance of the websocket
var webSocket = new
WebSocket("ws://localhost:8080/zkwebsocket/echo/?dtid=" +
zk.$('$win').desktop.uuid);

// receive a message from websocket, and notify ZK application to
update the component data.
webSocket.onmessage = function(event) {
    zkbind.$('$win').command('update'); // the update command has
already declared in ZKWebSocketViewModel.java
};

```

MVC Example

Here is the MVC way to send a command from client to server.

```
// Create a new instance of the websocket
var webSocket = new
WebSocket("ws://localhost:8080/zkwebsocket/echo/?dtid=" +
zk.$('$win').desktop.uuid);

// receive a message from websocket, and notify ZK application to
update the component data.
webSocket.onmessage = function(event) {
    zkservice.$('$win').command('update'); // the update command has
already declared in ZKWebSocketComposer.java
};
```

Command Parameter Converter

MVVM Example

When a user triggers a command with some data from client to server, the data should be in a Map (or says Object) type. For example,

```
zkbind.$('$win').command('update', {foo: 'myfoo', bar: {title:
'myBarTitle'}});
```

In the Java code

```
public static class Bar {
    private String title;
    public void setTitle(String title) { this.title = title; }
    public String getTitle() { return title; }
}
@Command("update")
@NotifyChange("count")
public void doUpdate(@ContextParam(ContextType.DESKTOP) Desktop
desktop, @BindingParam("foo") String myfoo, @BindingParam("bar") Bar
mybar) {
    count = desktop.<Integer>getStorage().getItem("count");
}
```

As you can see above, the data will automatically be converted into a specific object type according to the method declaration.

Note: developer can implement a custom Converter^[5] and specify it into the ZK library properties^[6].

MVC Example

When a user triggers a command with some data from client to server, the data should be in an array type in order.

For example,

```
zkservice.$('$win').command('update', [{foo: "myfoo"}, {bar: "mybar"}]); // the arguments should be in order within an array.¶
```

In the Java code

```
...  
    public static class MyFoo {  
        private String foo;  
        public void setFoo(String foo) { this.foo = foo; }  
        public String getFoo() { return this.foo; }  
    }  
  
    public static class MyBar {  
        // omitted  
    }  
  
    @Command  
    public void update(MyFoo foo, MyBar bar) {  
    }  
}
```

As you can see above, the data will automatically be converted into a specific object type according to the method declaration.

Note: developer can implement a custom Converter [7] and specify it into the ZK library properties [8].

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/Storage.html#>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/http/ZKWebSocket.html#>
- [3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/http/ZKWebSocket.html#getDesktopStorage\(javax.websocket.Session\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/http/ZKWebSocket.html#getDesktopStorage(javax.websocket.Session))
- [4] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/annotation/Command.html#>
- [5] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/bind/Converter.html#>
- [6] http://books.zkoss.org/wiki/ZK_Configuration_Reference/zk.xml/The_Library_Properties/org.zkoss.bind.jsonBindingParamConverter.class
- [7] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/Converter.html#>
- [8] http://books.zkoss.org/wiki/ZK_Configuration_Reference/zk.xml/The_Library_Properties/org.zkoss.zk.ui.jsonServiceParamConverter.class

Performance Tips

This chapter describes the tips to make your ZK application running faster. For information about identifying the bottleneck, please refer to the Performance Monitoring section.

For MVVM performance tips, please refer to MVVM Reference ^[1]

References

[1] <http://books.zkoss.org/zk-mvvm-book/8.0/advanced/performance-tips.html>

Use Compiled Java Codes

Not to Use zscript for Better Performance

It is convenient to use zscript in ZUML, but it comes with a price: slower performance. The degradation varies from one application to another. It is suggested to use zscript only for fast prototyping, POC, or small projects. For large websites, it is suggested to use ZK MVC/ZK MVVM ^[1] instead. For example, with ZK MVC

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

You can handle all events and components in `foo.MyComposer`. By the use of auto-wiring, it is straightforward to handle events and components.

Event Handler Is zscript

In addition to the `zscript` element, the event handler declared in a ZUL page is also interpreted at the runtime. For example,

```
<button label="OK" onClick="doSomething()"/>
```

where `doSomething()` is interpreted as `zscript`. Thus, for better performance, they should be replaced too.

Turn off the use of zscript

If you decide not to use zscript at all, you could turn on the disable-script configuration as follows, such that an exception will be thrown if zscript is used.

```
<system-config>  
    <disable-zscript>true</disable-zscript>  
</system-config>
```

Use the deferred Attribute

If you still need to write zscript codes, you can specify the `deferred` attribute to defer the evaluation of zscript codes as follows.

```
<zscript deferred="true">
</zscript>
```

By specifying the `deferred` attribute, the zscript codes it contains will not be evaluated when ZK renders a page. It means that the interpreter won't be loaded when ZK renders a page. This saves memory and speeds up page rendering.

In the following example, the interpreter is loaded only when the button is clicked:

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

The deferred Attribute and the onCreate Event

It is worth to notice that, if the `onCreate` event listener is written in zscript, the deferred option mentioned in the previous section becomes *useless*. It is because the `onCreate` event is sent when the page is loaded. In other words, all deferred zscript will be evaluated when the page is loaded if the `onCreate` event listener is written in zscript as shown below.

```
<window onCreate="init ()">
```

Rather, it is better to rewrite it as

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

Then, prepare `MyWindow.java` as shown below.

```
package my;
public class MyWindow extends Window {
    public void onCreate() { //to process the onCreate event
    ...
}
```

If you prefer to do the initialization right after the component (and all its children) is created, you can implement the `AfterCompose`^[2] interface as shown below. Note: the `afterCompose` method of the `AfterCompose` interface is evaluated at the Component Creation phase, while the `onCreate` event is evaluated in the Event Processing Phase.

```
package my;
public class MyWindow extends Window implements
org.zkoss.zk.ui.ext.AfterCompose {
    public void afterCompose() { //to initialize the window
    ...
}
```

Use the forward Attribute

To simplify the event flow, ZK components usually send the events to the component itself, rather than the parent or other targets. For example, when a user clicks a button, the `onClick` event is sent to the button. However, developers may need to forward the event to the window component by the use of the `onClick` event listener as follows.

```
<window id="w" onOK='alert("on OK")'>
    <button label="OK" onClick='Events.postEvent("onOK", w, null)' />
</window>
```

As suggested in the previous sections, the performance can be improved by *not* using zscript at all. Thus, you can rewrite the above code snippet either with `EventListener` or by specifying the `forward` attribute as follows.

```
<window apply="foo.MyComposer">
    <button label="OK" forward="onOK"/>
</window>
```

References

- [1] <http://books.zkoss.org/zk-mvvm-book/8.0/index.html>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/ext/AfterCompose.html#>

Use Native Namespace instead of XHTML Namespace

ZK creates a component (one of the derives of `AbstractTag`^[1]) for each XML element specified with the XHTML component set. In other words, ZK will maintain their states on the server. However, if you won't change their states dynamically (i.e., after instantiated), you could use the native namespace instead.

For example, the following code snippet creates five components (one `Table`^[2], `Tr`^[3], `Textbox`^[4] and two `Td`^[5]).

```
<h:table xmlns:h="xhtml">
    <h:tr>
        <h:td>Name</h:td>
        <h:td>
            <textbox/>
        </h:td>
    </h:tr>
</h:table>
```

On the other hand, the following code snippet won't create components for any elements specified with the native space (with prefix `n:`)^[6].

```
<n:table xmlns:n="native">
    <n:tr>
        <n:td>Name</n:td>
        <n:td>
            <textbox/>
        </n:td>
    </n:tr>
```

```
</n:table>
```

Notice that `table`, `tr` and `td` are generated directly to the client, so they have no counterpart at the server. You cannot change their states dynamically. For example, the following code snippet is incorrect.

```
<n:ul id="x" xmlns:n="native"/>
<button label="add" onClick="new Li().setParent(x)"/>
```

If you have to change them dynamically, you still have to use the XHTML component set, or you could use `Html`^[7] alternatively, if the HTML tags won't contain any ZUL component.

Notice that you could create the native components in Java too. For more information, please refer to the native namespace section.

```
HtmlNativeComponent n =
    new HtmlNativeComponent("table", "<tr><td>When:</td><td>", "</td></tr>");
n.setDynamicProperty("border", "1");
n.setDynamicProperty("width", "100%");
n.appendChild(new Datebox());
parent.appendChild(n);
```

-
- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zhtml/AbstractTag.html#>
 - [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zhtml/Table.html#>
 - [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zhtml/Tr.html#>
 - [4] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Textbox.html#>
 - [5] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zhtml/Td.html#>
 - [6] In fact, it will still create some components for the rerender purpose, such as `AbstractTag` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zhtml/AbstractTag.html#>). However, since they shall not be accessed, you could imagine them as not created at all.
 - [7] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Html.html#>

The Stub-izing of Native Components

By default, a native component will be *stub-ized*, i.e., they will be replaced with a stateless component called `StubComponent` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/StubComponent.html#>), such that the memory footprint will be minimized^[1]

Though rarely, you could disable the stubing by setting a component attribute called `org.zkoss.zk.ui.stub.native` (i.e., `Attributes.STUB_NATIVE` (http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/Attributes.html#STUB_NATIVE)). A typical case is that suppose you have a component that has a native descendant, and you'd like to detach it and re-attach later. Then, you have to set this attribute to false, since the server does not maintain the states of stub-ized components (thus, it cannot be restored when attached back).

```
<div>
    <custom-attributes org.zkoss.zk.ui.stub.native="false"/>
    <n:table xmlns:n="native"> <!-- won't be stub-ized -->
    ...

```

Once set, descendant components unless it was set explicitly.

[1] Non-native components could be stub-ized too by use of Table (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zhtml/Table.html#>). For more information, please refer here.

Version History

Version	Date	Content
5.0.6	January, 2011	The attribute called <code>org.zkoss.zk.ui.stub.native</code> was introduced to disable the <i>stub-ization</i> .

Use ZK JSP Tags instead of ZK Filter

The ZK filter actually maps each HTML tag to the corresponding XHTML components. As described in the previous section, it consumes more memory than necessary since ZK has to maintain the states of all ZK components (including XUL and XHTML components).

Include ZUL pages in a JSP page

If some part of UI is made of HTML tags (such as header and banner), you could use JSP as the main page, implement the parts with dynamic content in ZUL, and then put them together with `<jsp:include>`. For example,

```
<%-- main.jsp --%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>
</head>
<body>

<%-- the static part such as header --%>
<div>any content you like</div>

<%-- include the dynamic part --%>
<jsp:include page="foo.zul"/>

...
</body>
</head>
```

Use ZK components directly in a JSP page with ZK JSP tags

If you prefer to use ZK components directly in a JSP page, you could use ZK JSP tags ^[18]. For example,

```
<%-- another.jsp --%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<%@ taglib uri="http://www.zkoss.org/jsp/zul" prefix="z" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<z:zkhead />
```

```
</head>
<body>
<%-- any JSP content --%>

<z:page>
  <table>
    <tr>
      <td>Name</td>
      <td><z:textbox/></td>
    </tr>
  </table>
</z:page>
</body>
</head>
```

where `z:page` declares a ZK page and then ZK tags can be used inside it.

The above example is equivalent to the following code snippet, if a ZUL page is used,

```
<!-- another.zul -->
<?page complete="true"?>
<n:html xmlns="http://www.w3.org/1999/xhtml" xmlns:n="http://www.zkoss.org/2005/zk/native">
  <n:head>
  </n:head>
  <n:body>
    <%-- any HTML content --%>

    <n:table>
      <n:tr>
        <n:td>Name</n:td>
        <n:td><textbox/></n:td>
      </n:tr>
    </n:table>
  </n:body>
</n:html>
```

where `<?page complete="true"?>` declares that this page is a *complete* page, i.e., it will provide HTML's html, head and body tags as shown above.

Defer the Creation of Child Components

For sophisticated pages, the performance can be improved if we defer the creation of child components until they become visible. The simplest way to do this is by the use of the `fulfill` attribute. In the following example, the children of the second tab panel are created only if it becomes visible.

```
<tabbox>
  <tabs>
    <tab label="Preload" selected="true"/>
    <tab id="tab2" label="OnDemand"/>
  </tabs>
  <tabpanels>
    <tabpanel>
      This panel is pre-loaded since no fulfill specified
    </tabpanel>
    <tabpanel fulfill="self.linkedTab.onSelect">
      This panel is loaded only if tab2 receives the
      onSelect event
    </tabpanel>
  </tabpanels>
</tabbox>
```

For more information, please refer to the On-demand Evaluation section.

Defer the Rendering of Client Widgets

In addition to Defer the Creation of Child Components, you can defer the rendering of the widgets at the client by the use of the `renderdefer` attribute. It is a technique to make a sophisticated page appear earlier.

For example, we can defer the rendering of the inner window for 100 milliseconds as shown below

```
<window title="Render Defer" border="normal">
  The following is rendered after 100 milliseconds.
  <window title="inner" width="300px" height="200px" border="normal"
  renderdefer="100">
    Enter something <datebox onChange='i.value = self.value + " "'/>
    <separator/>
    <label id="i"/>
    <separator bar="true"/>
    <button label="say hi" onClick='alert("Hi")'>
  </window>
</window>
```

Unlike the `fulfill` attribute, the components on the server and the widgets at the client are created no matter if `renderdefer` is specified. It only defers the rendering of the widgets into DOM elements.

Here is another example in pure Java.

```
Tabpanel tp = new Tabpanel();
tp.setRenderdefer(0);
```

The render-defer technique is useful to improve the response time of showing a sophisticated page in a slow client. The total time required to render is not reduced (since all widgets still have to render later), but it allows the page to show up sooner and it makes the user feel more responsive.

Things you need to be careful about

[1] When you use this function on a child component, it's necessary to consider how the final style is shown after finishing "renderdefer". For example,

```
<zk xmlns:w="client">
<style>
    .always-scroll .z-grid-body {
        overflow-y: scroll !important;
    }
</style>
<grid sclass="always-scroll" width="400px" height="200px" >
    <columns>
        <column label="renderdefer (scrollbar col / odd row style missing) hflex col" hflex="1"></column>
    </columns>
    <rows>
        <row renderdefer="200"><cell height="50px">CELL 1</cell></row>
        <row renderdefer="400"><cell height="50px">CELL 2</cell></row>
        <row renderdefer="600"><cell height="50px">CELL 3</cell></row>
        <row renderdefer="800"><cell height="50px">CELL 4</cell></row>
        <row renderdefer="1000"><cell height="50px">CELL 4</cell></row>
    </rows>
</grid>
```

When you put "renderdefer" into row or rows without pre-defining to forcibly show the vertical-scrollbar, ZK treats it as the "zeroth" rows at first while hflex uses the total width of the grid without subtracting the width of the scrollbar. After "renderdefer" completes, we will find that the horizontal scroll bar has also appeared. However, this shouldn't be the case, only the y-scrollbar should have appeared. The solution to solve this is to force the vertical-scrollbar to show through CSS to prevent this from happening.

We recommend that you should defer the component by putting the "renderdefer" property on the component tag instead of putting it on children components (e.g. grid, columns, rows).

References

[1] <http://tracker.zkoss.org/browse/ZK-2336>

Client Render on Demand

- Available for ZK:



With Enterprise Edition, widgets^[1] will delay the rendering of DOM elements until really required. For example, the DOM elements of `comboitem` won't be created until the drop down is shown. It improves the performance a lot for a sophisticated user interface.

This feature is transparent to the application developers. All widgets are still instantiated (though DOM elements might not), so they can be accessed without knowing if this feature is turned on.

[1] A widget is the (JavaScript) object running at the client to represent a component

Client ROD: Tree

A tree node only renders its children node's DOM elements when it's open. If you close the node, it will remove those DOM elements. Thus, to have the best performance (particularly for a huge tree), it is better to make all tree item closed initially. You can observe this behavior with developer tool.

```
<treeitem forEach="${data}" open="false">
    <treerow>
        <treetcell label="${each.name}" />
        <treetcell label="${each.description}" />
    </treerow>
    <treetchildren>
        <treeitem forEach="${each.detail}" open="false">
            <treerow>
                <treetcell label="${each.name}" />
                <treetcell label="${each.description}" />
            </treerow>
            <treetchildren>
                <treeitem forEach="${each.fine}" open="false">
                    <treerow>
                        <treetcell label="${each.name}" />
                        <treetcell label="${each.description}" />
                    </treerow>
                </treeitem>
            </treetchildren>
        </treeitem>
    </treetchildren>
</treeitem>
```

Client ROD: Groupbox

Client ROD is enabled only if a groupbox is closed. Thus, to have the best performance (particularly with sophisticated content), it is better to make the groupbox closed initially if proper.

Client ROD: Panel

Client ROD is enabled only if a panel is closed. Thus, to have the best performance (particularly with sophisticated content), it is better to make the panel closed initially if proper.

Client ROD: Tabbox

Client ROD is enabled for the invisible tabpanels. After the tabpanel becomes active, its content will be rendered and attached to the DOM tree.

Client ROD: Organigram

Client ROD is enabled only if an orgitem is closed. Thus, to have the best performance (particularly with sophisticated content), it is better to make the orgitem closed initially if proper.

Enable or Disable Client ROD

If you want to disable Client ROD for the whole application, you can specify a library property called `org.zkoss.zul.client.rod` (https://www.zkoss.org/wiki/ZK_Configuration_Reference/zk.xml/The_Library.Properties/org.zkoss.zul.client.rod) with false. For example, specify the following in zk.xml:

```
<library-property>
    <name>org.zkoss.zul.client.rod</name>
    <value>false</value>
</library-property>
```

Or, if you prefer to disable it for a particular page, then specify false to a page's attribute called `org.zkoss.zul.client.rod`, such as

```
<custom-attributes org.zkoss.zul.client.rod="false" scope="page"/>
```

Or, if you prefer to disable it for all descendants of a particular component, then specify false to a component's attribute. And, you can enable it for a subset of the descendants. For example,

```
<window>
    <custom-attributes org.zkoss.zul.client.rod="false"/> <!-- disable it for descendants of window -->
    <div>
        <custom-attributes org.zkoss.zul.client.rod="true"/> <!-- enable it for descendants of div -->
    ..
    </div>
</window>
```

Listbox, Grid and Tree for Huge Data

In this section, we will discuss how to make a listbox, grid and tree to serve huge amount of data effectively.

Use Live Data and Paging

Sending out a listbox/grid/tree with a lot of items to the client is expensive. In addition, the JavaScript engines of some browsers are not good for initializing a listbox/grid/tree with a lot of items. A better solution is to use the live data, i.e., by assigning a model (such as ListModel^[7]) to it. Then, the items are sent to the client only if they become visible.

In addition, the performance will be improved more if you also use the paging mold such as

```
<listbox model="${mymodel}" mold="paging">  
...
```

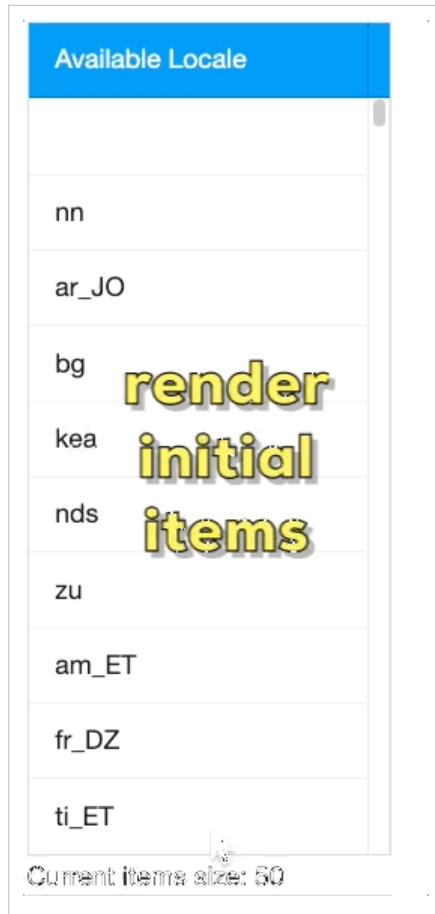
For more information of using and implementing a model, please refer to the Model section and ZK Component Reference: Listbox.

Turn on Render on Demand

- Available for ZK:
- CE PE EE

How ROD Improves Performance

With ZK EE, you can enable **Render on Demand** for Grid, Listbox, or Tree to boost performance when showing a huge amount of data. Grid and Listbox will load only the necessary data chunk from associated ListModel, render required Row(s)/Listitem(s) on the server, then create only the required corresponding widgets and render the DOM elements really needed in the browser. It improves the performance and saves memory significantly on both the server and browser sides.



Prerequisite

If you want to fully leverage the power of ROD, using a ZK model object like `ListModel` is necessary.

ROD actually brings a performance boost on both the client-side and server-side. However, if you use `forEach` to populate Rows or Listitems, the components will be all in memory, which does not give you any performance benefits on the server-side. (The client-side still enjoys a boost.)

Notice that you can enable this feature in different scopes.

Grid

If you want to enable Grid ROD for the whole application, you can specify a library property called Grid^[3] rod with `true`. For example, specify the following in zk.xml:

```
<library-property>
    <name>org.zkoss.zul.grid.rod</name>
    <value>true</value>
</library-property>
```

Or, if you prefer to enable it for a particular page, then specify `true` to a page's attribute called Grid^[3] rod, such as

```
<custom-attributes org.zkoss.zul.grid.rod="true" scope="page"/>
```

Or, if you prefer to enable it for all descendant grids of a particular component, then specify `true` to the component's attribute. You can enable it for a subset of the descendant grids. For example,

```
<window>
    <custom-attributes org.zkoss.zul.grid.rod="true"/> <!-- enable it for descendant grids of window -->
    <grid ...>
        ...
    </grid>
    <div>
        <custom-attributes org.zkoss.zul.grid.rod="false"/> <!-- disable it for descendant grids of div -->
        <grid ...>
            ...
        </grid>
        ...
    </div>
</window>
```

Fixed Viewport is Required

Note that Grid ROD will not work unless the Grid is configured with a limited **viewport**, so the user can see only a portion of rows; i.e. you have to set one of the following attributes:

- height
- vflex
- mold="paging"
- visibleRows

Specifies the number of rows rendered

```
[default: 100]
[inherit: true]
```

Specifies the minimum number of rows rendered on the client. It is only considered if Grid is using live data (Grid.setModel(ListModel)^[1]) and not using paging mold (Grid.getPagingChild())^[2].

```
<custom-attributes org.zkoss.zul.grid.initRodSize="30"/>
```

Listbox

If you want to enable Listbox ROD for the whole application, you can specify a library property called Listbox^[3] rod with true. For example, specify the following in zk.xml:

```
<library-property>
    <name>org.zkoss.zul.listbox.rod</name>
    <value>true</value>
</library-property>
```

Or, if you prefer to enable it for a particular page, then specify true to a page's attribute called Listbox^[3] rod, such as

```
<custom-attributes org.zkoss.zul.listbox.rod="true" scope="page"/>
```

Or, if you prefer to enable it for all descendant listboxs of a particular component, then specify true to the component's attribute. And, you can enable it for a subset of the descendant listboxs. For example,

```
<window>
    <custom-attributes org.zkoss.zul.listbox.rod="true"/> <!-- enable it for descendant listboxes of window -->
    <listbox ...>
        ...
    </listbox>
    <div>
        <custom-attributes org.zkoss.zul.listbox.rod="false"/> <!-- disable it for descendant listboxes of div -->
        <listbox ...>
            ...
        </listbox>
        ...
    </div>
</window>
```

Fixed Viewport is Required

Note that Listbox ROD will not work unless the Listbox is configured with a limited **viewport**, so the user can see only a portion of listitems; i.e. you have to set one of the following attributes:

- height
- vflex
- rows
- mold="paging"

Specifies the number of items rendered

```
[default: 100]
[inherit: true]
```

Specifies the number of items rendered when the Listbox first renders. It is used only if live data (Listbox.setModel(ListModel)^[3]) and not paging (<mold="paging">).

```
<custom-attributes org.zkoss.zul.listbox.initRodSize="30"/>
```

Tree

To turn off ROD for a tree, you need to specify org.zkoss.zul.tree.initRodSize with -1:

```
<custom-attributes org.zkoss.zul.tree.initRodSize="-1"/>
```

Limitation

`hflex="min"` has a not-resizing limitation, so if you specify it on a `<grid>/<column>` (`<listbox>/<listheader>`, or `<tree>/<treecol>`), a component doesn't resize the column when you scrolls down to see a wider item. Because the wider item is not rendered initially for render-on-demand. At the moment of calculating the width, the component doesn't count the wider item into width calculation.

Grid and Listbox will enlarge their columns when you scroll down and render a wider item.

References

- [1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Grid.html#setModel\(ListModel\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Grid.html#setModel(ListModel))
- [2] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Grid.html#getPagingChild\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Grid.html#getPagingChild())
- [3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Listbox.html#setModel\(ListModel\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Listbox.html#setModel(ListModel))

Implement ListModel and TreeModel

The default implementation of models, such as ListModelList^[1] and DefaultTreeModel^[2] assumes all data are available in the memory. It is not practical if a model has a lot of data. For huge amount of data, it is suggested to implement your own model by loading and caching only one portion of data at a time.

To implement your own model for a component it is recommended that you extend the correct abstract model type. For a ListModel extend from AbstractListModel^[6], for a GroupsModel extend AbstractGroupsModel^[3] and for a TreeModel extend AbstractTreeModel^[5] as described in the Model section. To implement a model that supports sorting, you have to implement Sortable^[2] too. Each time a user requires sorting, boolean) Sortable.sort(java.util.Comparator, boolean)^[3] will be called and the implementation usually clears the cache and re-generates the SQL statement accordingly.

Here is some pseudocode for a custom ListModel:

```
public class FooListModel extends AbstractListModel implements Sortable {
    private int _size = -1;
    private Object[] _cache;
    private int _beginOffset;
    private String _orderBy;
    private boolean _ascending, _descending;
    private Comparator _sorting;

    public int getSize() {
        if (_size < 0)
            _size = /*SELECT COUNT(*) FROM ...*/
        return _size;
    }

    public Object getElementAt(int index) {
        if (_cache == null || index < _beginOffset || index >= _beginOffset + _cache.length) {
            loadToCache(index, 100); //SELECT ... FROM .... OFFSET index
            LIMIT 100
                //if _ascending, ORDER BY _orderBy ASC
                //if _descending, ORDER BY _orderBy DSC
        }
        return _cache[index - _beginOffset];
    }

    @Override
    public void sort(Comparator cmpr, boolean ascending) {
        _cache = null; //purge cache
        _size = -1; //so size will be reloaded
        _descending = !_ascending = ascending;
        _orderBy = ((FieldComparator) cmpr).getRawOrderBy();
    }
}
```

```

    _sorting = cmpr;
    //Here we assume sort="auto(fieldName)" is specified in
ZUML, so cmpr is FieldComparator
    //On other hand, if you specifies your own comparator,
such as sortAscending="${mycmpf}",
    //then, cmpf will be the comparator you assigned
    fireEvent(ListDataEvent.CONTENT_CHANGED, -1, -1);
}

@Override
public String getSortDirection(Comparator cmpf) {
    if (Objects.equals(_sorting, cmpf))
        return _ascending ? "ascending" : "descending";
    return "natural";
}
}

```

The implementation of boolean) Sortable.sort(java.util.Comparator, boolean) [3] generally has to purge the cache, store the sorting direction and field, and then fire ListDataEvent.CONTENT_CHANGED [2] to reload the content.

The field to sort against has to be retrieved from the given comparator. If you specify "auto(fieldName)" to Listheader.setSort(java.lang.String) [3], then the comparator is an instance of FieldComparator [17], and you could retrieve the field's name from FieldComparator.getRawOrderBy() [21].

If you'd like to use your own comparator, you have to carry the information in it and then retrieve it back when boolean) Sortable.sort(java.util.Comparator, boolean) [3] is called.

Also notice that we cache the size to improve the performance, since ListModel.getSize() [4] might be called multiple times.

Here is some pseudo for a custom TreeModel which renders an Apache Commons VFS FileObject to be able to browse a filesystem:

```

public class VfsTreeModel extends AbstractTreeModel<FileObject> {
    public VfsTreeModel(FileObject root) {
        super(root);
    }

    @Override
    public FileObject getChild(FileObject parent, int index) {
        FileObject child = null;
        try {
            FileObject[] children = parent.getChildren();
            child = children[index];
        } catch (FileSystemException e) {
            throw new IllegalArgumentException(e);
        }
        return child;
    }

    @Override
    public int getChildCount(FileObject node) {

```

```
int childCount = 0;
try {
    FileType type = node.getType();
    if( type == FileType.FOLDER ){
        childCount = node.getChildren().length;
    }
} catch (FileSystemException e) {
    throw new IllegalArgumentException(e);
}
return childCount;
}

@Override
public boolean isLeaf(FileObject node) {
    boolean isLeaf = false;
    try {
        FileType type = node.getType();
        isLeaf = (type == FileType.FILE );
    } catch (FileSystemException e) {
        throw new IllegalArgumentException(e);
    }
    return isLeaf;
}
/*
 * Return the sibling index of each node in walk down from the
root.
*/
@Override
public int[] getPath(FileObject node) {
    List<Integer> paths = new ArrayList<Integer>();
    try {
        // walk upwards to root getting sibling index of each
child in each parent
        FileObject parent = node.getParent();
        while (parent != null &&
parent.getType().equals(FileType.FOLDER)) {
            FileObject[] children = parent.getChildren();
            for( int index = 0; index < children.length; index++) {
                FileObject c = children[index];
                if( node.equals(c)){
                    paths.add(index);
                    break;
                }
            }
            node = parent;
            parent = node.getParent();
        }
    }
```

```
        } catch (FileSystemException e) {
            throw new IllegalArgumentException(e);
        }
        int[] p = new int[paths.size()];
        for( int index = 0; index < paths.size(); index++) {
            p[index] = paths.get(p.length - 1 - index); // reverse
        }
        return p;
    }
}
```

When many treerows are open and further rows are expanded the re-rendering of the tree may visit many of the open rows. It is therefore recommended that you cache the results of any expensive calls where possible with a suitable eviction strategy.

For a real example, please refer to Small Talk: Handling sortable huge data using ZK and/or Small Talk: Handling huge data using ZK.

Version History

Version	Date	Content
6.0.0	02/03/2012	Sortable interface

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/ListModelList.html#>
- [2] http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/event/EventDataEvent.html#CONTENTS_CHANGED
- [3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Listheader.html#setSort\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Listheader.html#setSort(java.lang.String))

Minimize Number of JavaScript Files to Load

Overview

ZK loads the required JavaScript files only when necessary. It is similar to Java Virtual Machine's class loader, but ZK's JavaScript loader loads one JavaScript package at a time. It minimizes the number of bytes to be loaded to a browser. However, with an Internet connection, a Web page is loaded faster if the number of files to load is smaller (assuming the total number of bytes to transmit is the same).

ZK, by default, loads both `zul` and `zul.wgt` packages when the `zk` package is loaded, since they are the most common packages a ZK page might use. A ZK page generally uses more packages than that, and you, as an application developer, can pack them together to minimize the number of JavaScript files.

Notice that the more packages you pack, the larger the file. It will then slow down the load time if some of the packages are not required. Thus, you should only pack the packages that will be required by most users.

Minimize the Number of JavaScript Files for a ZUL Page

Case Study: ZK Sandbox

In `index.zul` [1] of ZK Sandbox [2] there are about 15 JavaScript files that will be initially loaded:

```
* http://www.zkoss.org/zksandbox/zkau/web/947199ea/js/zk.wpd
* http://www.zkoss.org/zksandbox/zkau/web/947199ea/js/zul.lang.wpd
* http://www.zkoss.org/zksandbox/macros/category.js
* http://www.zkoss.org/zksandbox/zkau/web/_zv2010062914/js/zkmax.wpd
* http://www.zkoss.org/zksandbox/zkau/web/_zv2010062914/js/zul.wgt.wpd
* http://www.zkoss.org/zksandbox/zkau/web/_zv2010062914/js/zul.wpd
* http://www.zkoss.org/zksandbox/zkau/web/_zv2010062914/js/zul.utl.wpd
* http://www.zkoss.org/zksandbox/zkau/web/_zv2010062914/js/zul.layout.wpd
* http://www.zkoss.org/zksandbox/zkau/web/_zv2010062914/js/zul wnd.wpd
* http://www.zkoss.org/zksandbox/zkau/web/_zv2010062914/js/zul.tab.wpd
* http://www.zkoss.org/zksandbox/zkau/web/_zv2010062914/js/zul.inp.wpd
* http://www.zkoss.org/zksandbox/zkau/web/_zv2010062914/js/zul.box.wpd
* http://www.zkoss.org/zksandbox/zkau/web/_zv2010062914/js/zul.sel.wpd
* http://www.zkoss.org/zksandbox/zkau/web/_zv2010062914/js/zk fmt.wpd
* http://www.zkoss.org/zksandbox/zkau/web/_zv2010062914/js/zul.mesh.wpd
```

This means that the browser will trigger 15 requests to load the 15 JavaScript files. Even if each file is not too big, it still takes more time to connect to the server and download it. However, we can specify a DSP file to include several JavaScript into one and declare it at the top of the `index.zul`.

For example, `/macros/zksandbox.js.dsp`

```
<%@ page contentType="text/javascript; charset=UTF-8" %>
<%@ taglib uri="http://www.zkoss.org/dsp/web/core" prefix="c" %>
<%@ taglib uri="http://www.zkoss.org/dsp/zk/core" prefix="z" %>
${z:setCWRCacheControl()}
<c:include page="~/js/zk(fmt.wpd)" />
<c:include page="~/js/zul(mesh.wpd)" />
<c:include page="~/js/zul(util.wpd)" />
```

```
<c:include page="~/js/zul.layout.wpd"/>
<c:include page="~/js/zul wnd.wpd"/>
<c:include page="~/js/zul.tab.wpd"/>
<c:include page="~/js/zul.inp.wpd"/>
<c:include page="~/js/zul.box.wpd"/>
<c:include page="~/js/zul.sel.wpd"/>
<c:include page="/macros/category.js"/>
```

Note:

1. The included JavaScript files have their own sequence, so you cannot place them in randomly.
2. The *zk.wpd* is a ZK core JavaScript file hence you don't need to include it.
3. The *zul.lang.wpd* is an I18N message, so you don't need to include it.
4. In ZK 5.0.4 we introduced a new feature(#System-wide_Minimizing_the_Number_of_JavaScript_Files). However, since the release of this new feature the packages **zul**, **zul.wgt**, and **zkmax** will be merged automatically into the ZK package, so you don't specify them in the the *zksandbox.js.dsp* file.
5. `int) DspFns.setCacheControl(java.lang.String, int)`^[3] is used to set the Cache-Control and Expires headers to 24 hours, so the JavaScript file will be cached for a day.

index.zul

```
<?script type="text/javascript" src="/macros/zksandbox.js.dsp"?>
// omitted
```

System-wide Minimizing the Number of JavaScript Files

If a package is used by all your pages, you could configure it system wide by specifying the packages in the language add-on. Please refer to ZK Configuration Reference/zk.xml/The language-config Element for how to specify a language add-on.

For example, if the *zul wnd* package (Window^[4]) is required for all pages, then you could add the following to the language add-on.

```
<javascript package="zul wnd" merge="true"/>
```

Notice that you have to specify the merge attribute which indicates that the JavaScript code of the package will be loaded with the *zk* package. In other words, the *~/js/zk.wpd* will contain all the packages specified with the merge attribute.

Also notice that if you use several DSP/JSP files to load multiple packages in a file as described in the previous section, you generally don't specify them here. Otherwise, you will load the same package twice (though it is safe, it wastes time).

Note: If you merge several JavaScript files into your own lang-addon.xml, but some JavaScript files need to be counted on *zul.lang.wpd*, such as the package *zul.inp*, you can't include this package into your lang-addon.xml. (it will be fixed in ZK 5.0.5+)

Turn Off the Merging of JavaScript Packages

As described above, both `zul` and `zul.wgt` packages are merged into the `zk` package. If you prefer to load them separately, you could disable it by specifying the `ondemand` attribute as follows.

```
<javascript package="zul" ondemand="true"/>  
<javascript package="zul.wgt" ondemand="true"/>
```

Notice that all packages are default to load-on-demand, you rarely need to specify the `ondemand` attribute, unless you want to undo the package that has been specified with the `merge` attribute.

References

- [1] <http://zk1.svn.sourceforge.net/viewvc/zk1/releases/5.0.7/zksandbox/src/archive/index.zul?view=log>
- [2] <http://www.zkoss.org/zksandbox>
- [3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/fn/DspFns.html#setCacheControl\(java.lang.String,%20java.util.List\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/fn/DspFns.html#setCacheControl(java.lang.String,%20java.util.List))
- [4] <http://www.zkoss.org/javadoc/latest/jsdoc/zul/wnd/Window.html#>

Load JavaScript and CSS from Server Nearby

If some of the client machines are far away from the application server, we could set up a server near the clients to host ZK's JavaScript and CSS files, and then configure the application server to generate the URLs of JavaScript and CSS (and images it refers) from the server near the clients.



***Notice :** the ZK static resource server is an application server which you **deploy only official ZK library to**, not your whole application.

How-to

1. Implement the URLEncoder^[1]
2. Add **library-property** configuration to the *zk.xml*

Document :	ZK Configuration	Reference/zk.xml/The Properties/org.zkoss.web.servlet.http.URLEncoder.	Library
------------	------------------	--	---------

3. Host ZK static resource server

Following is a sample :

Configuration

```
<library-property>
    <name>org.zkoss.web.servlet.http.URLEncoder</name>
    <value>org.zkoss.test.TestEncoder</value> <!-- Where the Implementation Class is -->
</library-property>
```

Implementation

```
package org.zkoss.test;

import javax.servlet.ServletContext;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import org.zkoss.web.servlet.http.Encodes.URLEncoder;
public class TestEncoder implements URLEncoder {

    @Override
    public String encodeURL(ServletContext ctx, ServletRequest
request, ServletResponse response,
        String uri, URLEncoder defaultEncoder) throws Exception {
        if (isStaticResource(uri)) {
            return getResourceHost() + uri.replace("~./", "");
        } else {
            return defaultEncoder.encodeURL(ctx, request,
response, uri, defaultEncoder);
        }
    }
    /**
     * file .wcs : CSS File
     * file .wpd : Javascript File
     */
    private boolean isStaticResource(String url) {
        // zul.lang.wpd should not be a static resource
        return url.startsWith("~./") &&
!url.endsWith("zul.lang.wpd") && (url.endsWith(".wpd") || url.endsWith(".wcs"));
    }
}
```

```

    * Detect where the ip is / who login / what kind of resource
server
    *
    * @return the host name include protocol prefix. (Client will
retrieve resource from it)
    */
private String getResourceHost() {
    return "http://SomeWhereNearbyMe/DefaultContext/zkau/web/";
}

}

```

Hosting ZK Static Resource

1. Create a WAR with all ZK JARs only (without your application code)
2. configure web.xml as a normal ZK application
3. deploy such special **ZK WAR** to a server near your customer and add the URL to your implementation of URLEncoder.

Don't know how to deploy on server ? Please refer to Installation Guide.

References

[1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/web/servlet/http/Encodes/URLEncoder.html#>

Specify Stubonly for Client-only Components

Overview

- Available for ZK:
- **CE** **PE** **EE**

- Available for ZK:

- **CE** **PE** **EE**

It is common that the states of some components are not required to be maintained on the server. A typical example is that an application might use some components, such as hbox, for layout and won't access it again after rendered. To minimize the memory footprint, ZK supports a special property called `stubonly` (`Component.setStubonly(java.lang.String)` [1]). Once specified with `true`, its states won't be maintained on the server (and all states are maintained at the client). For example,

```

<hbox stubonly="true">
</hbox>

```

- Notice this feature is available since ZK 5.0.4 EE, and available in CE since ZK 6.0.0.

Values of Stubonly: true, false and inherit

The default value of the `stubonly` property is `inherit`. It means the value is the same as its parent's, if any, or `false`, if no parent at all. Thus, if a component's `stubonly` is specified with `true`, all its descendants are stub-only too, unless `false` is specified explicitly. For example, in the following snippet, only `textbox` is *not* stub-only, while `hbox`, `splitter`, `listbox`, `listitem` and `label` are all stub-only.

```
<hbox stubonly="true">
    a stub-only label
    <textbox stubonly="false"/>
    <splitter/>
    <listbox>
        <listitem label="also stubonly"/>
    </listbox>
</hbox>
```

Limitation of Stub Components

When a component is stub only, it will be replaced with a special component called a stub component (`StubComponent`^[2]) after rendered. In addition, the adjacent stub components might be merged to minimize the memory further. Thus, the application should not access the component again on the server, if it is specified as stub only.

Invalidation

While a stub component cannot be invalidated directly, it is safe to invalidate its parent. ZK will rerender all non-stub components and retain the states of stub components at the client. For example, in the following snippet, it is safe to click the `invalidate` button. From an end user's point of view, there is no difference whether `stubonly` is specified or not.

```
<window>
    <button label="self.parent.invalidate()"/>
    <vbox stubonly="true">
        stubonly <textbox/>
    </vbox>
</window>
```

It is a special case that paging and `stubonly` cannot be applied at the same time. For example,

```
<listbox mold="paging" pageSize="1" >
    <listitem>
        <listcell stubonly="true"/>
    </listitem>
    <listitem>
        <listcell />
    </listitem>
</listbox>
```

Although paging will invalidate `listbox` and its children, `stubonly` needs the referred widget in client side which is detached during paging and throws mounting error.

Detach and Reuse

The detailed information of a stub component is stored at the client. It is removed when the component is removed. Thus, you can't reuse a component if the component has some stub components and it is detached. For example, the following code won't work:

```
public class MyListener implements EventListener {
    private static Window win;
    public void onEvent(Event evt) throws Exception{
        if (win ==null) {
            win = new Window();
            win.setTitle("Hello!");
            win.setClosable(true);
            Label testLabel = new Label("My Label");
            testLabel.setStubonly("true");
            win.appendChild(testLabel);
        }
        win.setParent(evt.getTarget().getFellow("mainWindow"));
        win.doModal();
    }
}
```

In the above example^[3], `win` is reused but it also has a stub component (a label). When the window is closed, all the information at the client is removed (since `Window`'s `onClose()` method detaches the window). Thus, if the event is executed again, the client can restore the detailed information back.

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

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

[3] Please refer to ZK-1094 (<http://tracker.zkoss.org/browse/ZK-1094>) for a real case.

Event Handling

ZK will preserve all registered event listeners and handlers when converting a stub-only component to a stub component. In other words, the listener will be called if the corresponding event is fired. However, since the original component no longer exists, the event is fired in the most generic format: an instance of `Event` (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Event.html#>), rather than a derived class.

For example, in the following snippet, `org.zkoss.zk.ui.event.StubEvent:onStub` (<https://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/StubEvent.html>) will be generated to `System.out`.

```
<textbox stubonly="true"
onChange='System.out.println(event.getClass().getName() + ":" + event.getName())' />
```

In addition, the target (`Event.getTarget()` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Event.html#getTarget\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/event/Event.html#getTarget()))) is the stub component rather than the original one, `Textbox`.

Client-side Programming

The client-side widget of a component is the same no matter if it is stub only. Thus, the application can have the full control by registering the client side event listener, such as

```
<textbox stubonly="true" w:onchange="doSomething(this.value)" xmlns:w="client"/>
```

In other words, the stub-only components behave the same at the client.

Refer to Client Side Programming and ZK Client-side Reference: General Control for more information.

Version History

Version	Date	Content
6.0.2	June, 2012	Bug: stubonly doesn't work (http://tracker.zkoss.org/browse/ZK-1182), and change the event handle from origin event to StubEvent.

Reuse Desktops

By default, a desktop is purged when the user browses to another URI or refreshes the page. Thus, the user can have the most updated information. However, if a page takes too long to generate, you can provide a plugin so-called *desktop recycle*.

First, you implement the DesktopRecycle ^[1] interface to cache and reuse the desktops which are supposedly being removed. Second, specify the class in WEB-INF/zk.xml. For example, let us assume the class you implement is called foo.MyRecycle, then add the following to zk.xml

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

org.zkoss.zkmax.ui.util/DesktopRecycle

- Available for ZK:
- CE PE EE

ZK provides a default implementation, the DesktopRecycle ^[2] class, to simplify the use. You can use it directly or extend from it. By default, it caches all desktops for all URIs. You can extend it to limit to certain paths by overriding the shallRecycle method, or not to use desktops older than a particular time by overriding the shallReuse method.

For example, we can limit the URL to cache to "/long-op/*", and re-generate the page if it has been served for more than 5 minutes.

```
public class MyRecycle extends org.zkoss.zkmax.ui.util/DesktopRecycle {
    protected boolean shallCache(Desktop desktop, String path, int cause)
    {
        return path.startsWith("/long-op");
    }
}
```

```
protected boolean shallReuse(Desktop desktop, String path, int secElapsed) {
    return secElapsed >= 300;
}
```

Implement Your Own Desktop Recycle

It is straightforward to implement the `DesktopRecycle` [1] interface from scratch, if you prefer. The basic idea is to cache the desktop when the `beforeRemove` method is invoked, and to reuse the cached desktop when the `beforeService` method is called.

References

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

Control resource caching

Understanding ZK cache breaking by URL

ZK applications rely on a number of resource files loaded by the client. These resources are mainly JS and CSS packages, but can also include images or other binaries necessary on client side. To improve performance, default ZK applications will use the following caching strategy. Resources are served to the client via a URL containing a hash value of the ZK version. If the ZK version doesn't change, this hash value will stay identical between server restarts, sessions, and page loading. This allows clients to cache ZK resources, since the URL doesn't change between page loads, and thus reduce page initialization times on client side.

When the ZK version changes, the hash value contained in the resources URLs also changes, and thus breaks caching on client side. This is important, as ZK internal JavaScript files should only be used in their intended ZK version.

In effect, this means that internal ZK files will be cached unless the ZK version is changed. After version changes, the resources are downloaded and the new version is cached, replacing the outdated resources.

Loading resources through the ZK cache enabled URLs

Resources declared from the classpath, or from ZK ClassWebResources [1] folder will be delivered to the clients through the same URL patterns as the internal ZK resources. This can apply to resources declared globally in lang-addon, or locally in a page with a script or style tag.

Forcing a cache clear on non-ZK resources

Since clients may cache a specific version of resources declared this way, they will not update them unless forced to do so by a clear-cache, a forced refresh, or a ZK version change. A simple way to break the cache when a new version of your resources must be downloaded is to use the built-in hashed value in the resource URLs. As stated before, the value will change if the ZK version changes. However, it will also change if the webapp build id changes.

This can be done by creating a new WebApp class extending the default SimpleWebApp and overriding the default getBuild() method. The build returned can be generated automatically on webapp init to force the clients to update on every server restart. It could also be defined manually as a string, or retrieved from a database to only force a restart when the developer updates the version id.

Cache breaking with zk.xml property

The `org.zkoss.zk.ui.versionInfo.enabled` can be used with a STRING value. If a STRING is declared as the value, this string will be used as salt when generating the URL version hash. As a result, making a change to this property will cause a new hash to be generated, and can be used to force a cache break on all resources loaded through the version cache url. This includes all internal ZK resources, such as JS and CSS content from the ZK libraries.

More information is available here: [org.zkoss.zk.ui.versionInfo.enabled](#)

Examples

Generating a random number on startup

```
public class ForceResourcesUpdateWebapp extends SimpleWebApp {

    private String randomizer;

    @Override
    public void init(Object context, Configuration config) {
        super.init(context, config);
        randomizer = Long.toHexString(System.currentTimeMillis());
    }

    @Override
    public String getBuild() {
        return super.getBuild() + randomizer;
    }
}
```

Manually setting a version id

```
public class ForceResourcesUpdateWebapp extends SimpleWebApp {
    @Override
    public String getBuild() {
        return super.getBuild() + "1.1.0";
    }
}
```

Once the webapp class has been created, it must be declared in zk.xml such as:

```
<system-config>
    <web-app-class>foo.bar.ForceResourcesUpdateWebapp</web-app-class>
</system-config>
```

Code samples

The following samples are available in github

- Cache break on webapp restart [2]
- Cache break on build ID change [3]
- Custom webapp class declaration in zk.xml [4]

References

- [1] https://www.zkoss.org/wiki/ZK_Configuration_Reference/zk.xml/The_Library_Properties/org.zkoss.web.util.resource.dir
- [2] <https://github.com/zkoss/zkbooks/blob/master/developersreference/developersreference/src/main/java/org/zkoss/reference/developer/performance/controlcache/ForceResourcesUpdateOnRestartWebapp.java>
- [3] <https://github.com/zkoss/zkbooks/blob/master/developersreference/developersreference/src/main/java/org/zkoss/reference/developer/performance/controlcache/ForceResourcesUpdateOnBuildChangeWebapp.java>
- [4] <https://github.com/zkoss/zkbooks/blob/master/developersreference/developersreference/src/main/webapp/WEB-INF/zk.xml#L587>

Miscellaneous

Button: use the os mold if there are a lot of buttons

The `trendy` mold of a button provides a better and more consistent look, especially for Internet Explorer 6. Unfortunately, the browser (particularly, Internet Explorer) will be slowed down if there are a lot of buttons (with `trendy`) in the same page.

Notice that the default mold is `os` in ZK 5, while `trendy` in ZK 3.6.
Since 7.0.0 There is no difference between `os` and `trendy`.

The default mold can be changed easily. For example,

```
<library-property>
    <name>org.zkoss.zul.Button.mold</name>
    <value>trendy</value>
</library-property>
```

Refer to ZK Configuration Reference for more information.

Prolong the Period to Check Whether a File Is Modified

ZK caches the parsed result of a ZUML page and re-compiles it only if it is modified. In a production system, ZUML pages are rarely modified so you can prolong the period to check whether a page is modified by specifying `file-check-period` in `WEB-INF/zk.xml` as shown below. By default, it is 5 seconds.

```
<desktop-config>
    <file-check-period>600</file-check-period><!-- unit: seconds -->
</desktop-config>
```

Security Tips

This chapter describes how to make ZK applications secure. ZK is designed to provide enterprise-grade security. However, there are still several discussions worth to take a look. You can also refer to a detail report here (PDF).

Cross-site scripting

Overview

Cross-site scripting^[1] (XSS) is a type of computer security vulnerability typically found in web applications that enables malicious attackers to inject client-side scripts into web pages viewed by other users. Because HTML documents have a flat, serial structure that mixes control statements, formatting, and the actual content, any non-validated user-supplied data included in the resulting page without proper HTML encoding may lead to markup injection.

What ZK Encodes

All Input Components Block XSS

To prevent a XSS attack, ZK components encode any value that might be input by a user by escaping & and other unsafe characters. For example, the following statement is totally safe even if `any_value` contains a script like `<script>alert('xss')</script>`:

```
<textbox value="${any_value}" />
```

Attributes to Generate Texts

Label component's `value` and those attributes that generate texts into a page including `label`, `title`, `tooltiptext`, `placeholder`, `name`, `type`, and message like `createMessage`, `emptyMessage`. (ZK encodes them with `zUtil.encodeXML()` at client-side.)

Since ZK implicitly turns an EL expression like `${myMessage}` on a zul into a Label, so it's encoded, too.

What ZK Doesn't Encode

Components used to Generate HTML Directly

ZK provides several ways to write HTML tags in a zul, including ZK `<html>` component, native namespace, and `xhtml` components. Since their purpose is to allow you to write HTML tags directly, ZK doesn't encode them.

Comboitem's content

The Comboitem^[2]'s `content` attribute is designed to allow applications to generate HTML content directly. In other words, it is not encoded. In most cases we expect these values to come from the server-side. However, if your application takes user input as the `content` property, you will need to encode it properly. For example, if the value of `any_content`, in the following example, is generated directly without proper encoding, it may be vulnerable to XSS attacks.

```
<html>${any_content}</html>
```

The content is sanitized by default to avoid XSS attack. Please don't use JavaScript in the content.

Some methods of Clients

As the name says this utility allows more direct client-side access. Thus the methods don't encode the strings passed into them to allow formatting of the messages at the client-side, e.g.:

```
Clients.showNotification("Successfully processed: <br/>" +  
myTextbox.getValue());
```

When displaying user input using methods such as Clients.showBusy(java.lang.String)^[7], Clients.showNotification(java.lang.String)^[14], or anything similar; or when using Clients.evalJavaScript(java.lang.String)^[3] to dynamically concatenate JS code, user input should be escaped carefully.

Client-side Actions

The client-side action is not encoded and the options are interpreted as a JSON object. In most cases we expect the values to come from the server-side. However, if you allow end-users to specify them (not recommended), you should encode them by yourself.

Page Directive

All attributes of <?page?> are not encoded.

Sanitize User Input

As a framework, ZK tries to maintain a good balance between flexibility and default settings. Regarding attributes that are not escaped by default, application developers should use ZK XMLs.escapeXML(java.lang.String)^[4] or Apache Commons Lang's StringEscapeUtils^[5] to sanitize user input if you are taking user input as these attributes.

References

- [1] http://en.wikipedia.org/wiki/Cross-site_scripting
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Comboitem.html#>
- [3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#evalJavaScript\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#evalJavaScript(java.lang.String))
- [4] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xml/XMLs.html#escapeXML\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/xml/XMLs.html#escapeXML(java.lang.String))
- [5] <https://commons.apache.org/proper/commons-lang/javadocs/api-2.6/org/apache/commons/lang/StringEscapeUtils.html>

Block Request for Inaccessible Widgets

Non-existing Components are Safer than Invisible Ones

Users can easily access inaccessible elements (such as disabled or invisible ones) with a browser developer tool. For example, a hostile user can make an invisible button visible and then click it to trigger unexpected actions. Thus, it is recommended not to create an element if it is not supposed to be accessible. For example, the first statement is safer than the second one in the following example:

```
<button unless="${accessible}" />  
<button visible="${accessible}" />
```

Block with InaccessibleWidgetBlockService

- Available for ZK:
- CE PE EE

ZK provides the `InaccessibleWidgetBlockService`^[1] to block events sent from inaccessible widgets with a default set of rules. An inaccessible widget is defined as one that is either disabled, invisible, or read-only. It's important to note that these default rules may not apply to all use cases.

Before 10.0.0, the Inaccessible Widget Blocking Service is not enabled by default. Users need to enable it manually.

In ZK 10.0.0 and later, this blocking service is enabled by default to enhance security.

Limitation

This service is an additional filter to improve security but does not replace verifying roles and permissions in your business logic. Always verify access on the server side.

Apply the Default Blocking Service

To apply it to the whole application, just specify the following in `zk.xml` as follows:

```
<listener>  
    <listener-class>org.zkoss.zkmax.au.InaccessibleWidgetBlockService$DesktopInit</listener-class>  
</listener>
```

Then, each time a desktop is created, an instance of `InaccessibleWidgetBlockService` is added to the desktop to block the requests from the inaccessible widgets.

Default Blocking Rules

- Block all events from **disabled** and **invisible** components
- Block `onChange`, `onChanging`, `onSelect` of a **read-only** component
- **Don't** block `onOpen`

Supported Components

All invisible components are blocked. Some components are blocked when they are disabled/read-only, as follows:

Component
Button
A
Listbox
MenuItem
NavItem
Textbox
Tree
Intbox
Spinner
Doublebox
Decimalbox
Longbox
Doublespinner
Timepicker
Timebox
Checkbox
Datebox
Combobox
Chosenbox
Selectbox

Specify Events to Block

If you just want to block particular events, not all events. Then, you can specify a list of events in `zk.xml` like below to control the behavior of `InaccessibleWidgetBlockService`^[1]. For example,

```
<library-property>
    <name>org.zkoss.zkmax.au.IWBS.events</name>
    <value>onClick,onChange,onSelect</value>
</library-property>
```

Implement Your Own Blocking Rules

If you want to block a request for inaccessible widgets for the whole application or for a particular desktop, you can implement the AuService^[2] interface to filter out unwanted requests. The implementation of AuService is straightforward. For example, the following example blocks only onClick of Button:

```
public class MyBlockService implements org.zkoss.zk.au.AuService {  
    public boolean service(AuRequest request, boolean everError) {  
        final Component comp = request.getComponent();  
        return (comp instanceof Button) &&  
"onClick".equals(request.getCommand());  
        //true means block  
    }  
}
```

References

[1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zkmax/au/InaccessibleWidgetBlockService.html#>

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

Denial Of Service

Overview

From OWASP^[1]

The Denial of Service (DoS) attack is focused on making a resource (site, application, server) unavailable for the purpose it was designed. There are many ways to make a service unavailable for legitimate users by manipulating network packets, programming, logical, or resources handling vulnerabilities, among others. If a service receives a very large number of requests, it may stop providing service to legitimate users. Denial-of-service attacks significantly degrade service quality experienced by legitimate users. It introduces large response delays, excessive losses, and service interruptions, resulting in direct impact on availability.

While OWSAP recommends^[2] a few techniques developers can employ against DoS at application level, ZK as a framework provides two features to protect against DoS as described below

Notes

[1] https://www.owasp.org/index.php/Denial_of_Service

[2] https://www.owasp.org/index.php/Denial_of_Service

Limit how many resources can be consumed

As explained above DoS attacks cause server to overload and stop responding by requesting large number of resources, it is logical to set a limit on how many resources are allowed per session to prevent server overload. Below are two configurations in ZK that you can use to set such limits.

Limit number of desktops per session

You can configure **max-desktops-per-session** in zk.xml as shown below. It indicates maximum number of desktops per session that are allowed. A desktop represents an HTML page for a browser. In other words, this number controls the number of concurrent browser windows allowed per session.

```
<session-config>
    <max-desktops-per-session>a_number</max-desktops-per-session>
</session-config>
```

Note : A negative number means no limitation at all.

Limit number of requests per session

You can configure number of requests per session in zk.xml as shown below. It indicates the maximum number of concurrent requests per session that are allowed. Each time a user types an URL at the browser, it creates a request and the request ends after the response is sent to the browser. In other words, this number controls how many concurrent requests the same user can send.

```
<session-config>
    <max-requests-per-session>a_number</max-requests-per-session>
</session-config>
```

Note : A negative number means no limitation at all, but it is not recommended due to the possibility of the denial-of-service (DoS) attacks.

Prevent sending same request multiple times

If your application has a button starting a long running operation you could use `button.setAutodisable()` in Java or `autodisable="self"` in ZUL to prevent DoS resulting from repeated button onClicks which might hold up precious server resources. For more details on Button autodisable refer here (http://books.zkoss.org/wiki/ZK_Component_Reference/Essential_Components/Button#Autodisable)

In addition to this, each and every ZK ajax request carries an additional http header called **ZK-SID**. The purpose of this ZK-SID header is to differentiate between multiple Ajax requests. If the same ajax request is resent with the same ZK-SID then it is ignored. This ZK-SID header helps to reduce server load and hence DoS attack by sending too many similar/repeated requests.

Cross-site Request Forgery

Overview

From OWASP [1]

Cross-Site Request Forgery (CSRF) is an attack that tricks the victim into loading a page that contains a malicious request. It is malicious in the sense that it inherits the identity and privileges of the victim to perform an undesired function on the victim's behalf, like change the victim's e-mail address, home address, or password, or purchase something. CSRF attacks generally target functions that cause a state change on the server but can also be used to access sensitive data. For most sites, browsers will automatically include with such requests any credentials associated with the site, such as the user's session cookie, basic auth credentials, IP address, Windows domain credentials, etc. Therefore, if the user is currently authenticated to the site, the site will have no way to distinguish this from a legitimate user request.

In short, a successful CSRF attack uses a valid http request, but often with malicious data to cause unwanted and unintended results, which assumes a valid identity of end user by using above mentioned browser's way of handling user related information.

ZK and CSRF attack limitations

According to OWASP mentioned CSRF Limitations [2], in order to mount a successful CSRF attack several things have to happen:

1. The attacker must target either a site that doesn't check the referrer header (which is common) or a victim with a browser or plug-in that allows referer spoofing (which is rare).

This can be avoided by adding a servlet filter that checks if all request referrer and origin headers contain the appropriate values.

2. The attacker must find a form submission at the target site, or a URL that has side effects, that does something (e.g., transfers money, or changes the victim's e-mail address or password).

By design ZK is an Ajax solution. Because of this design generally no form submit nor specific URL request can cause side effects.

3. The attacker must determine the right values for all the form's or URL's inputs; if any of them are required to be secret authentication values or IDs that the attacker can't guess, the attack will fail.

ZK generates unique ids for html elements that represent ZK components on client side and these unique ids are checked on server side when data containing them is passed via ZK's Ajax mechanism. For successful CSRF attack, the attacker will have to guess all unique ids for those html elements while submitting the malicious request. If the html element ids are not the same as they were when page rendered then the data is considered invalid by ZK and request is rejected at server side automatically.

Also note that ZK will regenerate these ids if the components are re-rendered via page refresh or components are re-created again.

4. The attacker must lure the victim to a Web page with malicious code while the victim is logged into the target site.

This is more of a humane issue and depends on the end user. Application developers should raise the awareness about CSRF by documenting this in their application documentation which end users can refer to.

ZK Desktop ID as CSRF token

The general recommendation to prevent CSRF is to use **Synchronizer Token Pattern**. Generally, this is done by inserting a unique token usually referred as **CSRF token** in the generated HTML and checking it at the server side on form submission. ZK employs a similar technique in the form of **desktop ID**. Each requested URL in ZK web application associates a Desktop instance on the server side. Please refer to Desktop and Pages for more details on the concept of Desktop in ZK.

ZK desktop is discarded and re-created each time a new page is loaded in a browser or even the current page is refreshed. On each re-rendering of a page, a new automatically generated unique id is assigned to a desktop.

Once the page is loaded this desktop ID is carried via ZK Ajax mechanism and on each interaction, this unique desktop ID will be passed as Ajax request POST data.

ZK CSRF Protection Notes:

1. There is no One-to-One relation between Desktop id and HTTP Session id.
2. Desktop ID is unique per page per URL. Even the same URL across different browser tabs in the same browser instance will be assigned the unique desktop ID.
3. For successful CSRF attack, the attacker not only has to guess unique desktop ID but also each and every unique ids assigned to the HTML element for corresponding ZK widgets on the client side. If even one is not correct then the entire request is rejected at server side without executing any application level code containing business logic.

References

- [1] [https://www.owasp.org/index.php/Cross-Site_Request_Forgery_\(CSRF\)](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF))
[2] https://en.wikipedia.org/wiki/Cross-site_request_forgery#Limitations

OWASP Top 10 Security Concerns In 2017

What is the OWASP Top 10?

The Open Web Application Security Project (OWASP^[1]) is a worldwide not-for-profit charitable organization focused on improving the security of software. The OWASP Top 10^[2] is a powerful awareness document for web application security that presents a list of the 10 most critical web application security risks. The most recent edition of this document was published in 2017.

OWASP Top 10 in 2017

In the subsections that follow, we provide our statements against each of the top 10 security risks. Interested parties are encouraged to visit OWASP to see this document in full, or other abundant web resources for more information about each security risk. Depending on the nature of vulnerability, a front-end framework such as ZK is not the source of weaknesses that need to be strengthened. Application developers need to understand the vulnerabilities leading to the possible exploits attackers may choose to target your system. With that knowledge, software authors can take preventative measures to mitigate these threats.

(A1) Injection ^[3]

ZK has no assumption about any 3rd party technologies, and cannot cover their required escaping syntax. This security risk needs to be addressed during application development where untrusted data were utilized in conjunction with an interpreter. For example, to prevent SQL injection, user data should not be used to construct SQL command directly; instead, parameterized queries should be used.

(A2) Broken Authentication ^[4]

Since ZK does not provide any login mechanism, it is up to developers to choose and secure user authentication management mechanism on their own.

(A3) Sensitive Data Exposure ^[5]

Developers have full control over which data is displayed in a zul page, and must avoid exposing sensitive data. Internal resources should be stored in a non-webapp accessible location, such as below the WEB-INF folder.

(A4) XML External Entities (XXE) ^[6]

Since the framework main purpose is client-server communication inside a web page, ZK itself doesn't access XML based services or downstream integrations. It is up to the application developer to exercise judgement when implementing these sources if appropriate in their design. Since this treatment will be done in the business layer of the application, it is not impacted by ZK.

(A5) Broken Access Control ^[7]

As a layout and communication framework, ZK isn't concerned by access control. Access control should be handled by the developer. One way of handling it can be done at a lower level by leveraging an existing web application access and security framework such as Spring security. Since ZK server code uses Java, authentication and access tokens provided by the security framework can be used in the business layer to make access control decision.

(A6) Security Misconfiguration ^[8]

Security misconfiguration can happen at any level of an application stack, including the platform, web server, application server, database, framework, and custom code. Developers and system administrators need to work together to ensure that the entire stack is configured properly.

(A7) Cross-Site Scripting (XSS) ^[9]

Please see our tips ^[10] on how to deal with this security issue in ZK.

(A8) Insecure Deserialization ^[11]

ZK doesn't store states on the client. The client UI is a representation of the abstract page located on server side, which cannot be tampered with by the user. User actions trigger events listeners and values updates on the component used in this page. ZK components check for data consistency and will throw exceptions if an illegal request is made by the client such as trying to select a non-existent item in a list. However, the developer should consider all client content suspect by default and leverage ZK validators as well as implementing their own consistency checks to make sure that incoming data match expectations. The developer should also avoid storing state themselves on client side.

(A9) Using Components with Known Vulnerabilities ^[12]

ZK addresses known vulnerabilities at high priority. Once identified, we provide updates and patches as soon as possible. Hence, it is recommended to upgrade to the latest version when it becomes available.

(A10) Insufficient Logging & Monitoring ^[13]

ZK provides logging for Framework related actions, warnings, exceptions and errors covering topics ranging from resources loaded by the framework to illegal operations on ZK components. Logging relative to the business layer of an individual application should be implemented by the application developer. Since ZK server is Java based, developers can leverage any log infrastructure fulfilling their requirement, such as slf4j.

References

- [1] The Open Web Application Security Project official website (https://www.owasp.org/index.php/Main_Page)
- [2] The OSWAP Top 10 (https://www.owasp.org/index.php/Top_10-2017_Top_10)
- [3] https://www.owasp.org/index.php/Top_10-2017_A1-Injection
- [4] https://www.owasp.org/index.php/Top_10-2017_A2-Broken.Authentication
- [5] https://www.owasp.org/index.php/Top_10-2017_A3-Sensitive_Data_Exposure
- [6] [https://www.owasp.org/index.php/Top_10-2017_A4-XML_External_Entities_\(XXE\)](https://www.owasp.org/index.php/Top_10-2017_A4-XML_External_Entities_(XXE))
- [7] https://www.owasp.org/index.php/Top_10-2017_A5-Broken_Access_Control
- [8] https://www.owasp.org/index.php/Top_10-2017_A6-Security_Misconfiguration
- [9] [https://www.owasp.org/index.php/Top_10-2017_A7-Cross-Site_Scripting_\(XSS\)](https://www.owasp.org/index.php/Top_10-2017_A7-Cross-Site_Scripting_(XSS))
- [10] http://books.zkoss.org/wiki/ZK_Developer%27s_Reference/Security_Tips/Cross-site_scripting
- [11] https://www.owasp.org/index.php/Top_10-2017_A8-Insecure_Deserialization
- [12] https://www.owasp.org/index.php/Top_10-2017_A9-Using_Components_with_Known_Vulnerabilities
- [13] https://www.owasp.org/index.php/Top_10-2017_A10-InsufficientLogging%26Monitoring

Content Security Policy

What is Content security policy?

Content-security-policy (CSP) is a security standard introduced to prevent XSS attacks (cross-site scripting) and other content injection attacks.

To reduce those injection risks, CSP provides a way for web applications and website owners to declare permissions for loading scripts from only approved and trusted sources. To enable CSP, you can either configure your web server to return the CSP HTTP header, or use the <meta> element.

See more: Content Security Policy Level 2 ^[1]

How to use Content security policy?

To use CSP in your web application, the first thing you need to know is that not all the browsers support CSP.

(Support browsers: Content Security Policy 1.0 ^[2], Content Security Policy Level 2 ^[3])

To enable CSP, you can either configure your web server to return the CSP HTTP header, or use the <meta> element. The following "directives" are recommended to be defined, which is for protecting against XSS attacks. For complete information please reference CSP official documents ^[4].

1. default-src

The default-src is the default policy for loading content such as Javascript, CSS, fonts, etc.

2. script-src / style-src / img-src / font-src

Defines valid sources of JavaScript/stylesheets/images/fonts.

3. connect-src

Applies to AJAX, WebSocket or EventSource.

4. child-src

Governs the creation of nested browsing contexts as well as Worker execution contexts.

Examples

1. Only allows loading resources from the same origin.

```
default-src 'self';
```

2. Allows loading scripts from the same origin and Google Analytics.

```
script-src 'self' www.google-analytics.com;
```

Using Content Security Policy in ZK

We don't suggest applying too strict CSP to ZK, because internally ZK still needs to use some 'unsafe-eval' and 'unsafe-inline' declarations when loading scripts and CSS files. However, you can still use CSP in ZK to enhance supported parts and make your application safer than before. Here's an example of a relaxed policy used in a ZK application:

```
<?header name="Content-Security-Policy-Report-Only"
         value="default-src 'none';
                script-src 'self' 'unsafe-inline' 'unsafe-eval';
                frame-src 'self';
                connect-src 'self' ws://your.server.name:8080/;
                img-src 'self';
                style-src 'self' 'unsafe-inline';
                font-src 'self'" ?>
```

- Using <?header ?>^[5] to specify a response header.

References

- [1] <https://www.w3.org/TR/CSP2/>
- [2] <https://caniuse.com/#feat=contentsecuritypolicy>
- [3] <https://caniuse.com/#feat=contentsecuritypolicy2>
- [4] <https://www.w3.org/TR/CSP/>
- [5] https://www.zkoss.org/wiki/ZUML_Reference/ZUML/Processing_Instructions/header

SSO Redirect Handling

In this section, we assume you already know the basics of SSO (Single-Sign-On) flow like CAS web flow ^[1] or Active Directory Federation Services ^[2].

AJAX Request Gets 302 Redirect

Redirect including SSO (Single-Sign-On) handling has always been a common challenge in Ajax, and that's no exception when it comes to ZK. You may have run into this error:

```
The response could not be parsed: Expected JSON format (please check
console for details).
Unexpected token '<':
```

Or in an older ZK version:

```
The server is temporarily out of service.
Would you like to try again?

(Unexpected token < (SyntaxError))
```

It usually happens when:

- session timeout
- your access token is invalid for some reason

If you check developer tool > Network, you should see a 302 Redirect response on a ZK AU request:

Name	Status	Type	Initiator
zkau	302	fetch / Redirect	zk.wpd:28752
login?redirect=http%3A%2F%2Flocalhost%3A8080%2Fzkap...	202	preflight	Preflight ⓘ

If this happens, it's most likely you have a service that intercepts HTTP requests (e.g. a security filter) and redirects the AU request to a login page.

According to the HTTP specification, browsers will follow the 302 redirect to visit the target URL transparently. Browsers will receive the HTML content of the login page as the response to the AU request. However, ZK client engine expects a JSON format response for an AU request, not the HTML content, and therefore reporting the error.

Solution: Turn 302 to 403

Because of atomic HTTP redirect handling^[3], browsers handle redirecting transparently, it's not something we can change on the ZK side. What we can do is to turn the 302 into 403 so that we can handle it properly.

First, configure your SSO server or the security filter to return the response code **403 Forbidden** instead of 302 for the situation mentioned above (session expired or invalid access token).

Then, configure the error-reload Element, so that ZK can handle 403 by reloading the specified login page.

In some special setups, you might need to override javascript function `zAu._fetch()`. If you are not sure how to, please contact ZK support.

Library Customization Reference

Most SSO related frameworks/libraries provide customizable filters, here are some examples, please refer to the official and latest documents on their website:

Spring Security

- web security config for response 403^[4]
- error reload^[5]

Spring Reference Doc

- 10.9. AbstractAuthenticationProcessingFilter^[6]
- AuthenticationFailureHandler (javadocs)^[7]

Apache Shiro

- AccessControlFilter.onAccessDenied^[8]
- related stack overflow^[9]

CAS

- Java client docs^[10]
- implement a filter similar to ErrorRedirectFilter^[11]

OKTA

- customize Filter to use 403 instead of redirect^[12]

References

- [1] <https://apereo.github.io/cas/4.2.x/protocol/CAS-Protocol.html>
- [2] <https://docs.microsoft.com/en-us/archive/blogs/askds/understanding-the-ad-fs-2-0-proxy>
- [3] <https://fetch.spec.whatwg.org/#atomic-http-redirect-handling>
- [4] <https://github.com/zkoss/zkspringboot/blob/redirect302/zkspringboot-demos/zkspringboot-security-demo/src/main/java/org/zkoss/zkspringboot/security/WebSecurityConfig.java#L48-L51>
- [5] <https://github.com/zkoss/zkspringboot/blob/redirect302/zkspringboot-demos/zkspringboot-security-demo/src/main/resources/mbeaninfo/zk/zk.xml#L11-L15>
- [6] <https://docs.spring.io/spring-security/site/docs/5.4.1/reference/html5/#servlet-authentication-abstractprocessingfilter>
- [7] <https://docs.spring.io/spring-security/site/docs/5.4.1/api/org/springframework/security/web/authentication/AuthenticationFailureHandler.html>
- [8] [https://shiro.apache.org/static/1.6.0/apidocs/org/apache/shiro/web/filter/AccessControlFilter.html#onAccessDenied\(javax.servlet.ServletRequest,javax.servlet.ServletResponse\)](https://shiro.apache.org/static/1.6.0/apidocs/org/apache/shiro/web/filter/AccessControlFilter.html#onAccessDenied(javax.servlet.ServletRequest,javax.servlet.ServletResponse))

- [9] <https://stackoverflow.com/questions/41709987/how-to-make-shiro-return-403-forbidden-with-spring-boot-rather-than-redirect-to>
- [10] <https://github.com/apereo/java-cas-client/>
- [11] <https://github.com/apereo/java-cas-client/blob/master/cas-client-core/src/main/java/org/jasig/cas/client/util/ErrorRedirectFilter.java>
- [12] <https://developer.okta.com/blog/2019/07/22/servlet-authentication>

Performance Monitoring

To improve the performance of an Ajax application, it is better to monitor the performance for identifying the bottleneck. Depending on the information you'd like to know, there are a few approaches.

- PerformanceMeter^[1]: Monitoring the performance from the network speed, server-processing time and the client-rendering time.
- EventInterceptor^[2]: Monitoring the performance of each event listener.
- Monitor^[3]: Monitoring the number of desktops, sessions and other system load.
- There are a lot of performance monitor tools, such as VisualVM^[4] and JProfiler^[5]. They can provide more insightful view of your application.

For sample implementations, you might take a look at the following articles:

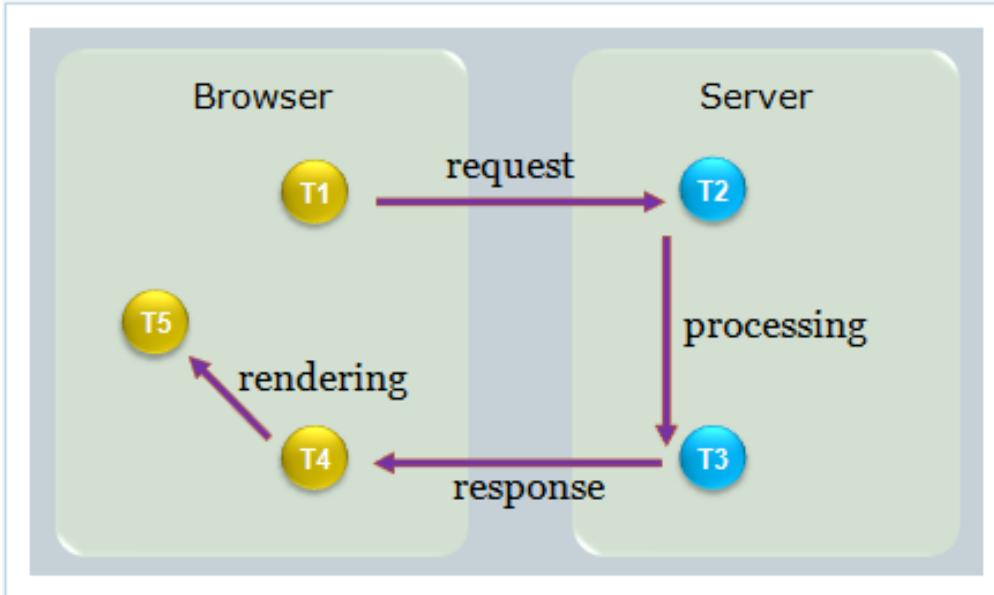
- Performance Monitoring of ZK Applicaiton
- A ZK Performance Monitor
- Real-time Performance Monitoring of Ajax Event Handlers

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/PerformanceMeter.html#>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/EventInterceptor.html#>
- [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Monitor.html#>
- [4] <https://visualvm.github.io/>
- [5] <http://www.ej-technologies.com/products/jprofiler/overview.html>

Performance Meters

PerformanceMeter^[1] is a collection of callbacks that the implementation could know when a request is sent, arrives or is processed.



As shown above, T1-T5 identifies the following callbacks.

- T1: (java.lang.String, org.zkoss.zk.ui.Execution, long) PerformanceMeter.requestStartAtClient(java.lang.String, org.zkoss.zk.ui.Execution, long)^[1]
- T2: org.zkoss.zk.ui.Execution, long) PerformanceMeter.requestStartAtServer(java.lang.String, org.zkoss.zk.ui.Execution, long)^[2]
- T3: org.zkoss.zk.ui.Execution, long) PerformanceMeter.requestCompleteAtServer(java.lang.String, org.zkoss.zk.ui.Execution, long)^[3]
- T4: org.zkoss.zk.ui.Execution, long) PerformanceMeter.requestReceiveAtClient(java.lang.String, org.zkoss.zk.ui.Execution, long)^[4]
- T5: org.zkoss.zk.ui.Execution, long) PerformanceMeter.requestCompleteAtClient(java.lang.String, org.zkoss.zk.ui.Execution, long)^[5]

Thus,

- Server Execution Time: T3 - T2
- Client Execution Time: T5 - T4
- Network Latency Time: (T4 - T3) + (T2 - T1)

How it works

Notice that, when we make a connection to load a page for the first time, only Server Execution Time is available. T4 and T5 will be saved on the client-side and sent back along with the next request.

If you print the request ID and the method name, when zk calls a performance monitor. You will see a different log between loading a zul and sending an AU request.

Request a ZUL

If load a zul 3 times:

```
# first load zul
requestId1st - requestStartAtServer
requestId1st - requestCompleteAtServer

# 2nd load zul
requestId1st - requestReceiveAtClient
requestId1st - requestCompleteAtClient
requestId2nd - requestStartAtServer
requestId2nd - requestCompleteAtServer

# 3rd load zul
requestId2nd - requestReceiveAtClient
requestId2nd - requestCompleteAtClient
requestId3rd - requestStartAtServer
requestId3rd - requestCompleteAtServer
```

Send AU Requests

If you send 2 AU requests, you will see a log like:

```
# 1st au
requestId - requestReceiveAtClient
requestId - requestCompleteAtClient
requestId-0 - requestStartAtClient
requestId-0 - requestStartAtServer
requestId-0 - requestCompleteAtServer

# 2nd au
requestId-0 - requestReceiveAtClient
requestId-0 - requestCompleteAtClient
requestId-1 - requestStartAtClient
requestId-1 - requestStartAtServer
requestId-1 - requestCompleteAtServer
```

- only when sending au request, zk calls `requestStartAtClient()`

Register as a Listener

Once implemented, you need to register it as a listener in `WEB-INF/zk.xml` to make it work: (assume the class is called `foo.MyMeter`):

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

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/PerformanceMeter.html#requestStartAtClient>
- [2] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/PerformanceMeter.html#requestStartAtServer\(java.lang.String,java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/PerformanceMeter.html#requestStartAtServer(java.lang.String,java.lang.String))
- [3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/PerformanceMeter.html#requestCompleteAtServer\(java.lang.String,java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/PerformanceMeter.html#requestCompleteAtServer(java.lang.String,java.lang.String))
- [4] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/PerformanceMeter.html#requestReceiveAtClient\(java.lang.String,java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/PerformanceMeter.html#requestReceiveAtClient(java.lang.String,java.lang.String))
- [5] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/PerformanceMeter.html#requestCompleteAtClient\(java.lang.String,java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/PerformanceMeter.html#requestCompleteAtClient(java.lang.String,java.lang.String))

Event Interceptors

Though `EventInterceptor`^[2] is designed to allow developer to intercept how an event is processed, you could use it as callback to know how long it takes to process an event. The event processing time can be calculated by subtracting the time between `EventInterceptor.beforeProcessEvent(org.zkoss.zk.ui.event.Event)`^[1] and `EventInterceptor.afterProcessEvent(org.zkoss.zk.ui.event.Event)`^[2]

Once implemented, you could register it by specifying the following in `WEB-INF/zk.xml` (assume the class is called `foo.MyEventMeter`):

```
<zk>
  <listener>
    <listener-class>foo.MyEventMeter</listener-class>
  </listener>
</zk>
```

References

- [1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/EventInterceptor.html#beforeProcessEvent\(org.zkoss.zk.ui.event.Event\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/EventInterceptor.html#beforeProcessEvent(org.zkoss.zk.ui.event.Event))
- [2] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/EventInterceptor.html#afterProcessEvent\(org.zkoss.zk.ui.event.Event\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/EventInterceptor.html#afterProcessEvent(org.zkoss.zk.ui.event.Event))

Loading Monitors

To know the loading of an application, you could implement Monitor^[3] to count the number of desktops, sessions and requests.

Once implemented, you could register it by specifying the following in WEB-INF/zk.xml (assume the class is called foo.MyStatistic):

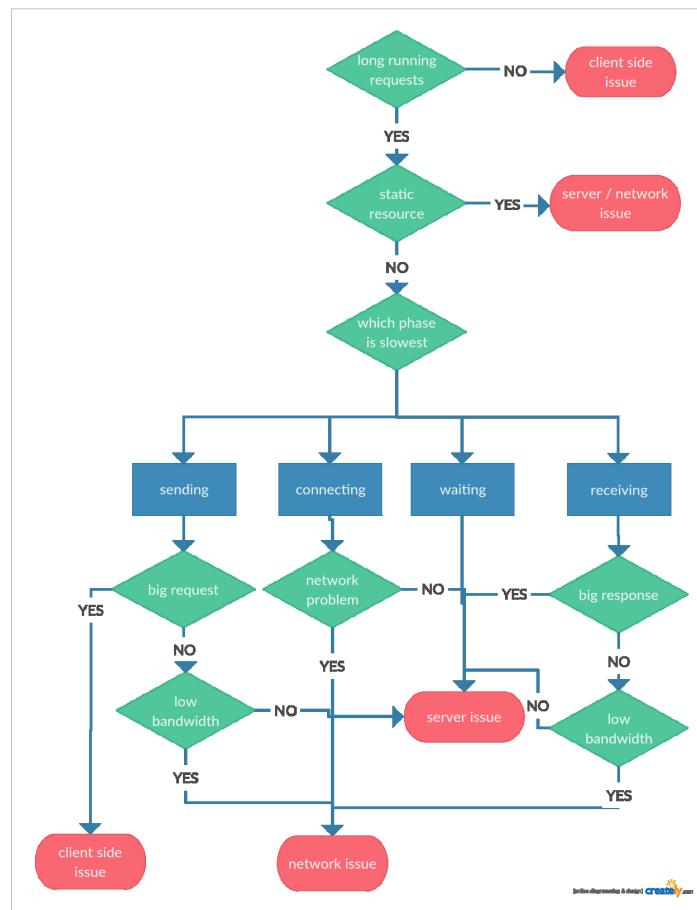
```
<zk>
  <listener>
    <listener-class>foo.MyStatistic</listener-class>
  </listener>
</zk>
```

Step by Step Trouble Shooting

Step by Step Trouble Shooting

This article tries to provide general steps and details on how to determine a bottleneck in a slow performing ZK application. Additionally, it offers some conclusions and tips, what the next steps would be after identifying the problem area.

A flow chart to summarize the whole trouble shooting process:



Identify the Bottleneck

Usually the bottleneck can be found in one of these areas:

- client
- network
- server

To breakdown a slow performing web application is a good idea to start, where the bad performance is perceived... in the browser. Most modern browsers provide very sophisticated tools supporting the search for a bottleneck and draw some conclusions, and eliminate other possible causes easily.

Developer tools - Net(network)

Chrome -> [F12] / [CTRL + SHIFT + I]

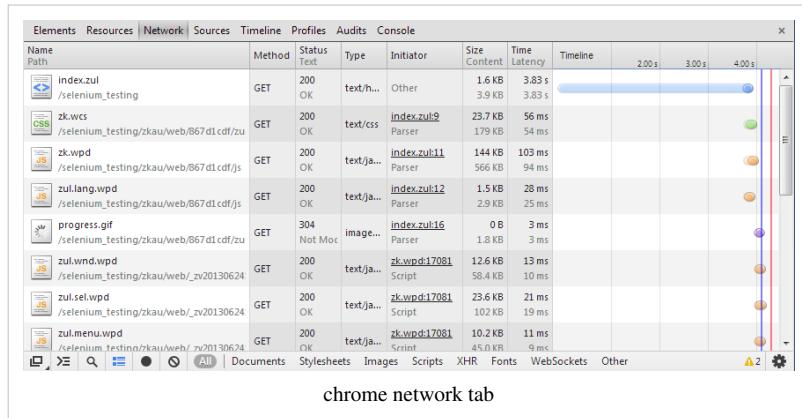
Firefox -> [CTRL + SHIFT + Q]

Firefox with Firebug -> [F12]

IE9+ -> [F12]

IE8 & others -> fiddler2

Investigating the network traffic by following the questions below the biggest problem area(s) should become apparent after a few minutes:



1. Are there one or more long running requests?

NO → #Client Side Issue

YES

↓

2. Is it a static resource?

YES (js, css, images...) → check #ZK Server Configuration (debug / cache / compression) → STILL SLOW → #Network Issue

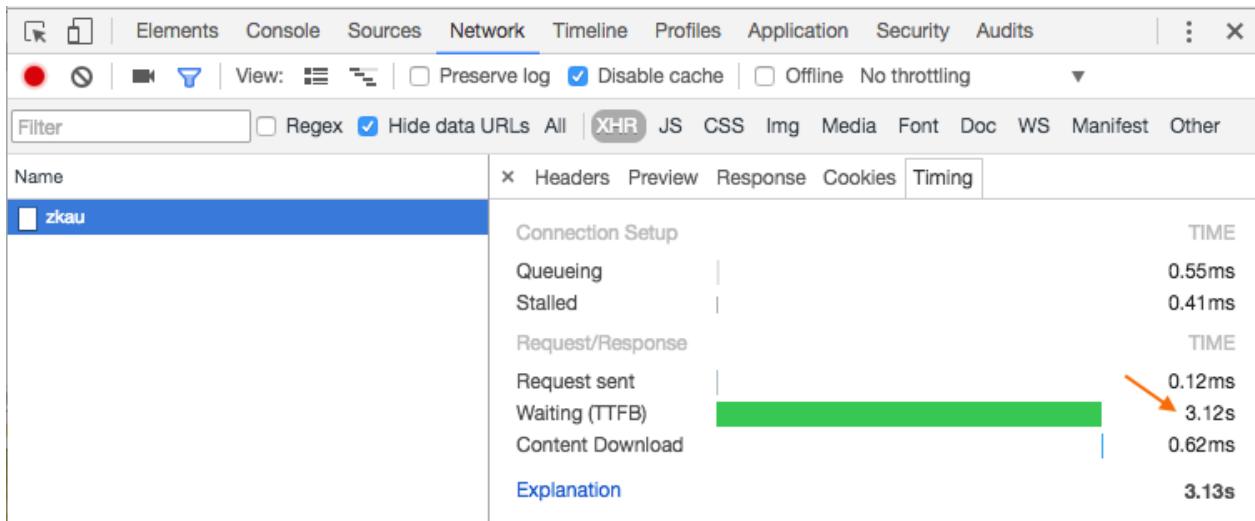
NO (dynamic request into ZK application)

- *.zul = full page request (can be followed by ajax requests)
- zkau/* = ajax request

↓

3. Which PHASE of the request is slowest ?

(wording based on Chrome developer tools - EDIT Chrome updated wording and explanations^[1])



CONNECTING (or one of Proxy, DNS Lookup, Blocking, SSL)

3.a) Is this a network problem (everything between browser and ZK Application)?

- test ping / trace route to different servers
- test dns lookup timing

YES → #Network Issue

NO → #Server Side Issue (application takes long time to accept connection, or even times out)

SENDING

3.b) Is the request unreasonably big? (rare case, usually due to an upload (reasonable), or form posting a lot of data)

YES → #Client Side Issue

NO

↓

3.c) Is the bandwidth low?

- e.g. try upload the same amount of data to the server via ftp/scp to check possible upload speed

YES → #Network Issue

NO → #Server Side Issue (application server receiving request data slowly)

WAITING → #Server Side Issue (application server taking long time to prepare response)

Content Download

3.d) Is the response unreasonably big?

YES → #ZK Server Configuration (render on demand / compression)

NO

↓

3.e) Is the bandwidth low?

- e.g. try to download the same amount of data from the server via ftp/scp to check download speed

YES → ask your administrator to fix it ;)

NO → #Server Side Issue (appserver sending response data slowly)

Client Side Issue

If there is no significant time spent on the Network and Server side, the slowdown must happen somewhere on the client side.

Client side performance is affected by many factors and may vary with different browser types / versions. Other factors are:

- operating system
- available memory
- CPU speed / load
- screen resolution
- graphics card speed

So it is good to compare the client performance on different computers with different browsers, to identify the configuration causing the issue.

If client performance among configurations / browsers is equally bad, the issue will more likely be found in the Rendering area → #Client Side Profiling

Once you identified the client side rendering takes very long, check the size of the response, if the Client engine needs to render a lot (e.g. a Grid with 1000 lines) it will take its time. So compare the timing with a smaller response, and consider if this can be prevented by reducing the data sent to the client using Render on Demand ^[2] or Pagination ^[3] (Most users don't need 1000 lines visible at once)

Performance degrading over time when using the application (while network and server timings remain constant) might indicate a client side memory leak → #Client Memory Issue

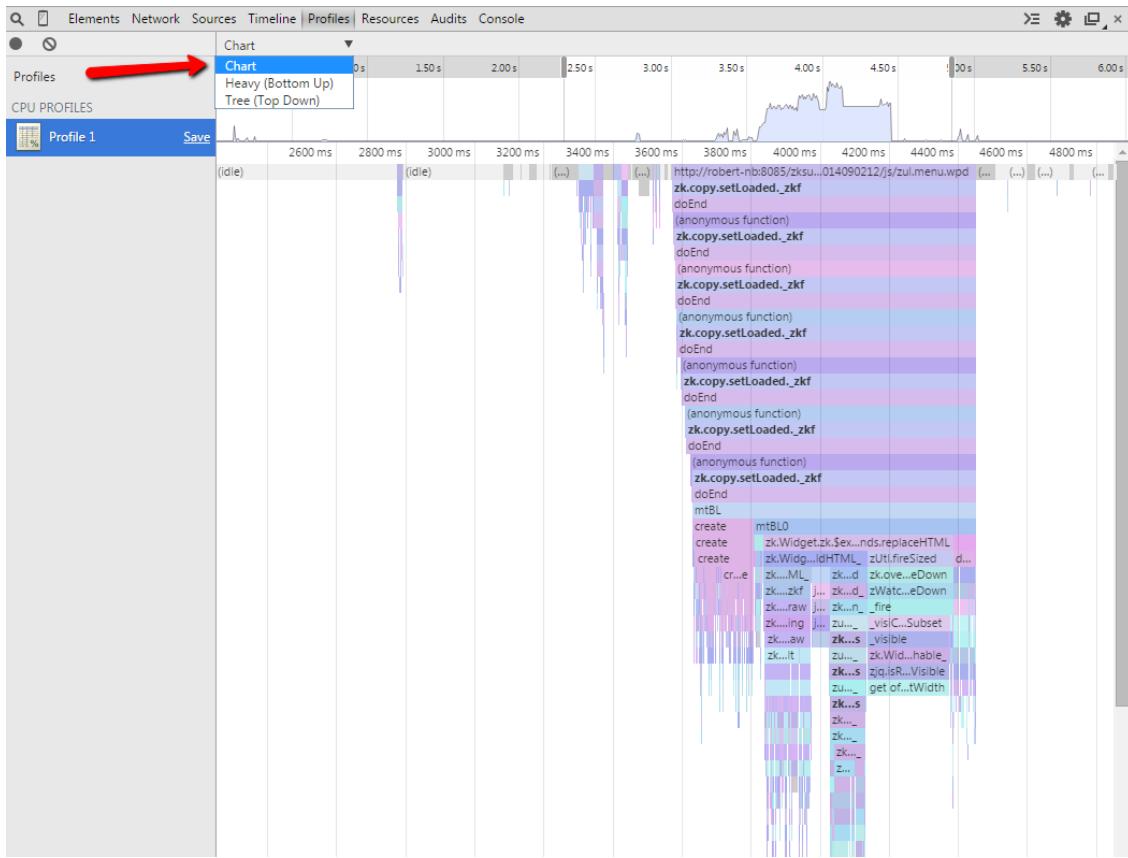
Client Side Profiling

Make sure your local computer is not under heavy CPU load, and has "enough" Memory available, before starting to profile the Javascript execution in the browser.

To measure and break down the time spent in the JS engine you can try the following steps (in chrome), and interpret the results:

1. switch to the "profiles"-tab
2. choose "Collect Javascript CPU Profile"
3. click "start"
4. perform your action e.g. reload the screen, or load the search results
5. click "stop"
6. switch to "Chart" (choice at the top)

You'll get a nice view like this: (**Update:** since chrome version 58 the "profiles" and "timeline" are combined in the "performance"-tab more... ^[4])

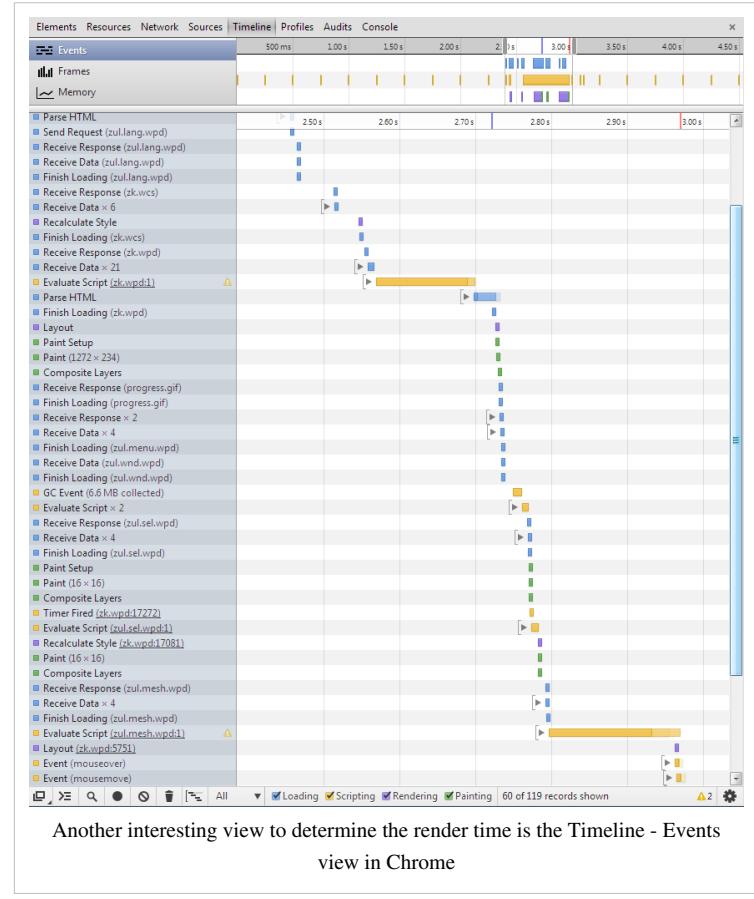


This brilliant visualization of the JS execution flow and stack depth can be used / interpreted in many ways to extract the information you require.

The timeline on the top indicates the whole period between "start" and "stop", I selected the range we are interested in, and the colorful area at the bottom gives details about which methods are actually called and their timing (you can zoom in and out using the mouse wheel too), clicking on one method will directly lead you to the associated line in the source code (enabling debug.js [5] will help when using this feature).

The small peak (at 2800ms) on the left side is my actual event refreshing the page. The gap until 3300ms represents the idle time of the JS engine waiting for the first response from the server, then you can follow in which order the JS files are executed and which of them consume most of the time.

Additional waiting times in the middle indicate load time of additional JS files and garbage collection times. At about 4450ms



the JS engine stops meaning ZK has finished rendering the page widgets and updating the DOM elements.

We can conclude our page took about 1650ms (after the initial event) to load and render in order to become available to the user and there is no significant slowdown on either JS or network side.

General conclusions:

- more wider mountains → mean more JS time (e.g. ZK render time, a third party library)
- more valleys (flat lines) → mean more waiting time (mostly network, maybe also CSS formatting or other time the browser does not assign to the JS engine)

A similar but less detailed and colorful view is available in Firefox [Shift + F5] showing some basic timeline. And in IE you can also get profiling data.

I find the flame-chart in chrome most powerful, and even if the Performance issue only occurs in a different browser it is a good starting point to visualize the timing of the problem, and understand the complexity of the execution path (e.g. in IE you get the information that most time is spent in method x(), and in chrome you can actually see the method in a bigger context). Or when using an older version of IE, you can define the methods of interest to put some log statements to trace performance manually.

Client Memory Issue

Use a system management tool (e.g. on windows: task manager or process explorer) to watch the memory consumption of your browser over time.

If memory is increasing when repeating an action on a page -> subsequently remove/add items from your ZUL file to identify the component causing this issue.

Again Chrome offers a memory timeline view, real time memory profiling functions and JS heap dumps.

Server Side Issue

After identifying the server side as the bottleneck we can drill down the problem even further.

There are many things on the server side, that can cause a slowdown in the response. Of course you'll first check that there is enough physical memory available and that no other (unrelated) process on the server causes CPU load while you are actually idling in the ZK application.

Basically ensure that the application server has all the resources it needs and is configured to use them:

e.g.

- CPUs (multi cores)
 - the server process might only have limited access to available CPU cores
- Memory
 - you might have a lot of physical memory but the JVM is configured to only use a small amount of that
- Incoming connections:
 - application server might be configured to handle only a small number of simultaneous requests even if it could handle more
 - queueing/denying additional requests exceeding these limits
- DB connection pools
 - there might be a very fast DB waiting for your input, but your connection pools are too small

Then perform the "slow" operation and observe CPU load, IO times and Memory.

Busy or Waiting?

If the CPU is almost idle but most time is spent in IO operations, then it could be a slow or very busy hard disk, or long running network operations (such as accessing external web services or querying a DB on a different host). The network or DB doesn't necessarily have to be slow, it could just be a large accumulating number of very quick calls to external resources.

E.g. 300 requests of 10ms (I would not consider slow) each, still take 3 seconds - during these 3 seconds your java process could be idly waiting for the responses and you'll see a low figure on CPU compared to a high value in IO wait times.

You'll most likely know which external resources you are accessing in your code, so commenting these out and temporarily replacing them with mock implementations will help you exclude these culprits. If you are not aware of any external resources: continue with Server Side Profiling to identify the places that take most time when creating the response.

Here are some helpful tools to determine busy or waiting processes:

Linux:

- top (in most cases just sufficient to distinguish between busy or waiting)

Windows:

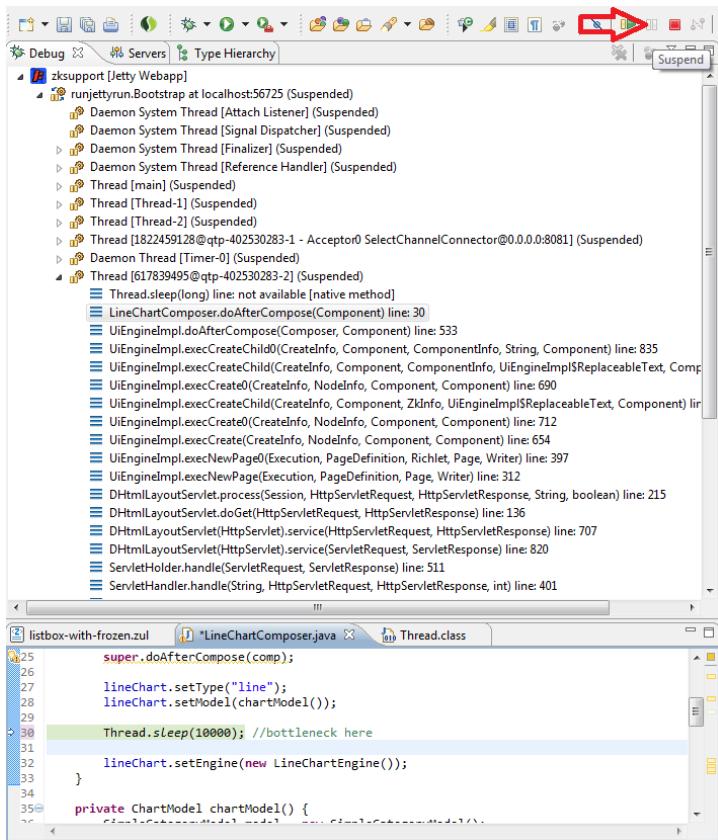
- Process explorer ^[6] (a better "task manager")
- Perfmon.exe (please check this tutorial ^[7] for more details)

Performance Debugging/Logging/Tracing

Feeling Lucky

Sometimes real profiling might be overkill or you just want to make a few quick probing tests. Then a simple way to find a long running method is to launch your server in debug mode without any break point. Then trigger the slow operation. Right in the middle of the operation, "Suspend" the execution (yes you can suspend manually without defining a break point), in your IDE. You'll have several suspended threads. Just examine their call stacks, and usually the longest stack is the one you are interested in. Then go down in the stack to search for the bottleneck (Chances are high, that you'll suspend the execution during the actual call causing the bottleneck - either it takes a long time to execute, or it is called very often).

In eclipse a very obvious case might look like this (just imaginarily replace Thread.sleep() with db.query(), url.openConnection(), webService.get() ...):



Not so lucky

In other cases the bottleneck will not be as outstanding as in the above image, but still the call stack can be a good source to start from. e.g. outputting some counters and tracing information to the log.

Putting extra tracing logs will not always be possible and also takes implementation, building, deployment time.

So if this is not quickly possible it is better to do some performance sampling or profiling.

Server Side Profiling

If the JavaVM is busy during the operation or you have no idea which parts in your application consume most of the time, **sampling** or **profiling** are the final weapons.

- Sampling: analyzes the call stacks of all threads at given intervals and summarizes statistics about which methods are active most of the time
 - sufficient in most cases, will find the biggest bottlenecks at a high chance
- Profiling: gives exact more detailed timing, but this is at a cost, you should already know what you are looking for before starting the profiler
 - profiling everything might just "kill" the application

There are several Profiling tools in various price ranges and there is JVisualVM (included in the JDK) so it should be available everywhere.

This is a nice tutorial about how to get started with JVisualVM : Profiling with VisualVM - Part 1 ^[8] and Part 2 ^[9]

Sampling example using JVisualVM

here is a small example showing 2 different kinds of time performance issues (a busy loop, and a suspended thread (sleeping)):

```
@Override
public void doAfterCompose(Component comp) throws Exception {
```

```
super.doAfterCompose(comp);

lineChart.setType("line");
lineChart.setModel(chartModel());

long start = System.currentTimeMillis();
while(System.currentTimeMillis() - start < 3000) { //bottleneck here
    StringBuilder builder = new StringBuilder();
    for(int i = 0; i < 100; i++) {
        builder.append(new String("XXX"));
    }

builder.toString().getBytes(Charset.forName("utf-8"));
}

lineChart.setEngine(new LineChartEngine());
}

private ChartModel chartModel() throws InterruptedException {
    Thread.sleep(5000); //another bottle neck here
    SimpleCategoryModel model = new SimpleCategoryModel();

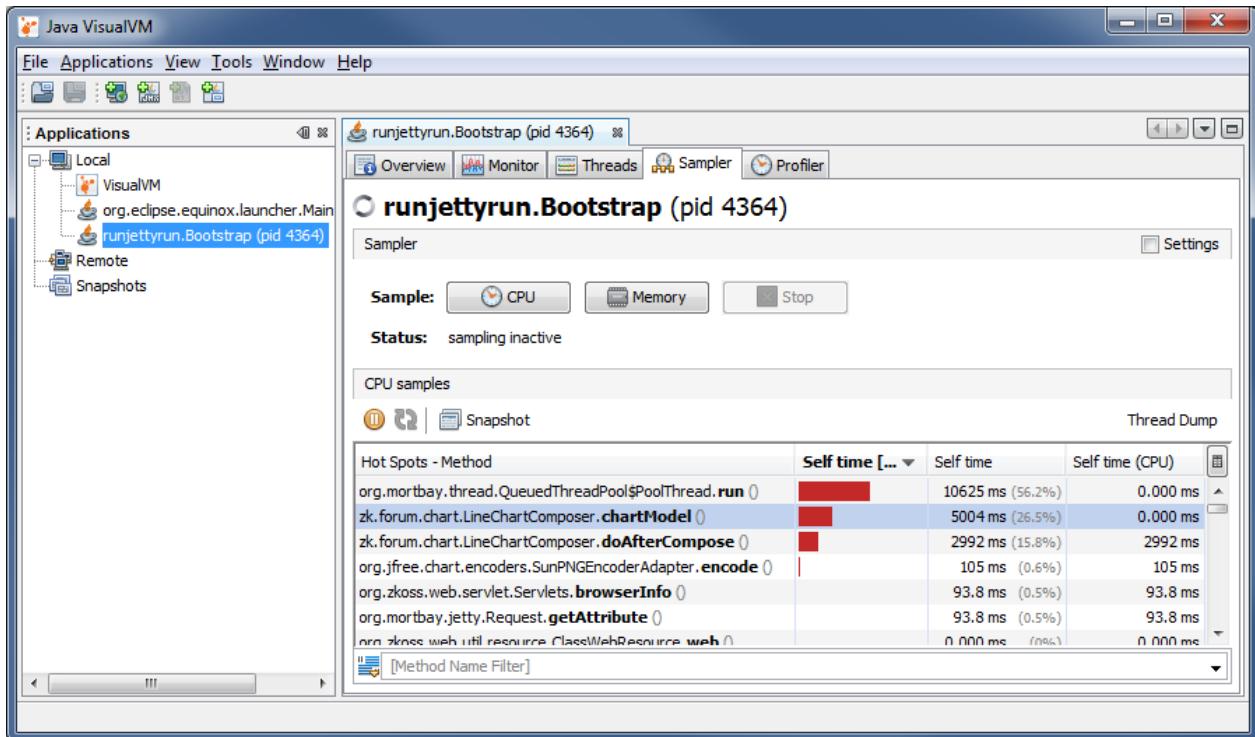
    Calendar date = Calendar.getInstance();

    for(int i = 0; i < 10; i++) {
        model.setValue("spent money", date.getTime(),
Math.random() * 10000);
        date.add(Calendar.DATE, 1);
    }

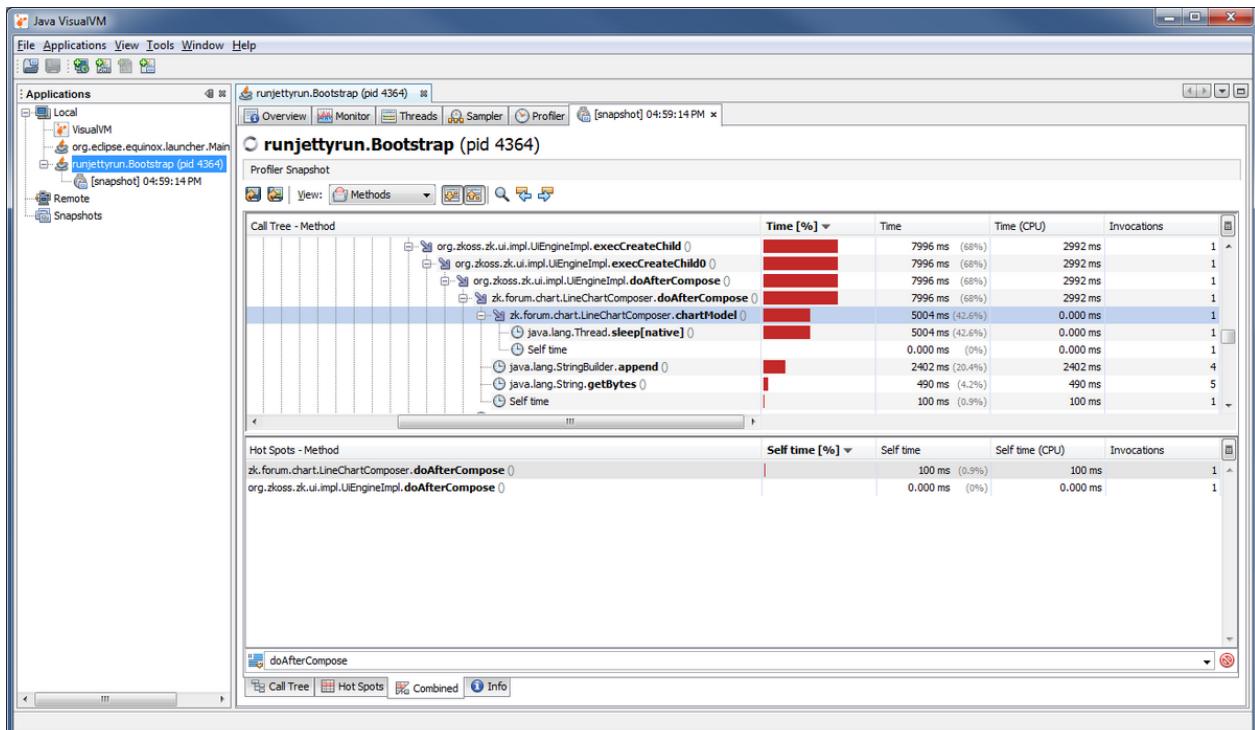
    return model;
}
```

When executing this slow page, it will delay the page rendering by ~8 seconds, where 3 seconds will be spent in a busy loop (causing a peak on the CPU) doing some useless string building, and 5 seconds you wouldn't notice on your CPU, as the thread is just sleeping.

Starting the sampler will show the actual "Hot Spots" like this.



The 2 slow methods appear, then just take a snapshot, and view the details about the actual call stack in the combined view, and filter by the method name.



In the call tree we actually see what is happening inside in more detail:

- the "chartModel()"-method is sleeping for 5 seconds
 - and the "doAfterCompose()"-method is performing time consuming String operations for roughly 3 seconds
- Performance issue located!

Memory Issue

Memory issues come in many different flavors:

The most common ones are explained here <http://plumbr.eu/blog/understanding-java-lang-outofmemoryerror>.

Ensure your physical memory, and the memory assigned to the JVM are sufficient.

Memory Leak

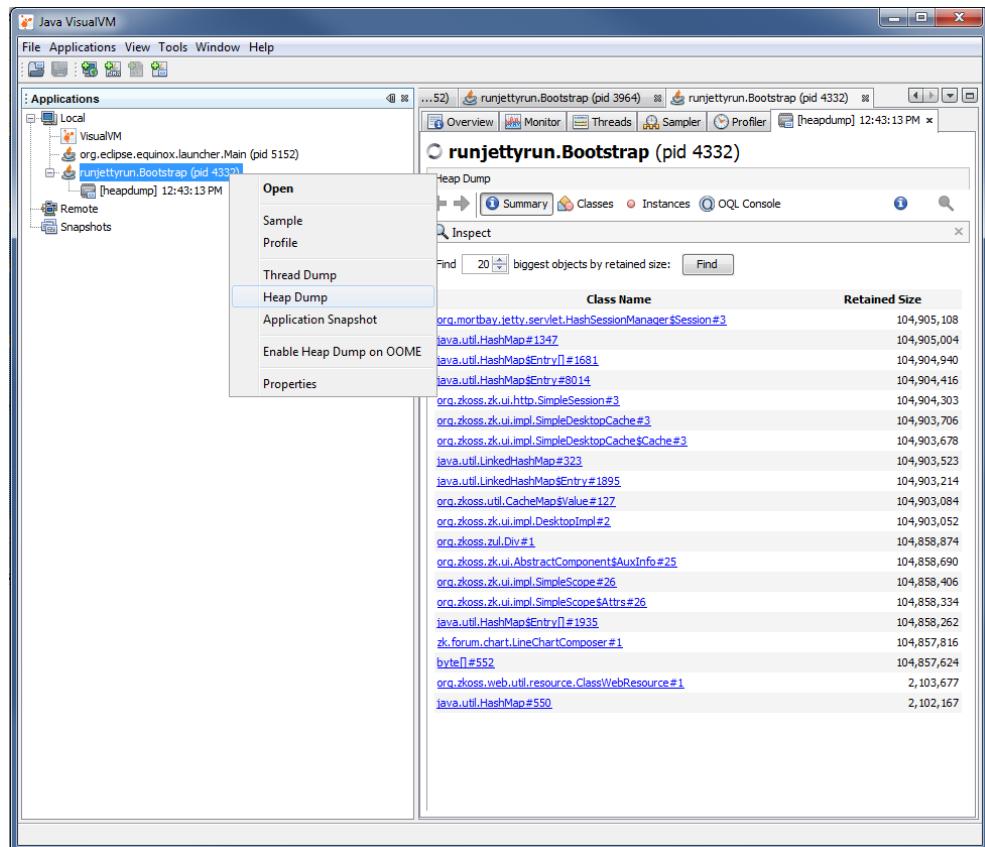
When running out of heap space, you'll either have a good explanation for it i.e. know that you allocate big lumps of data (and how often) or you might have a memory leak, indicated by your constantly increasing memory, and not being released again, after a Garbage Collection.

Here is a very simple case using JVisualVM. Take this Composer code snippet which allocates a lot of memory.

```
public class LineChartComposer extends SelectorComposer {
    @Wire("#linechart")
    private Chart lineChart;

    private byte[] bigMemoryBlock = new byte[100 * 1024 * 1024]; //100
MB
    ...
}
```

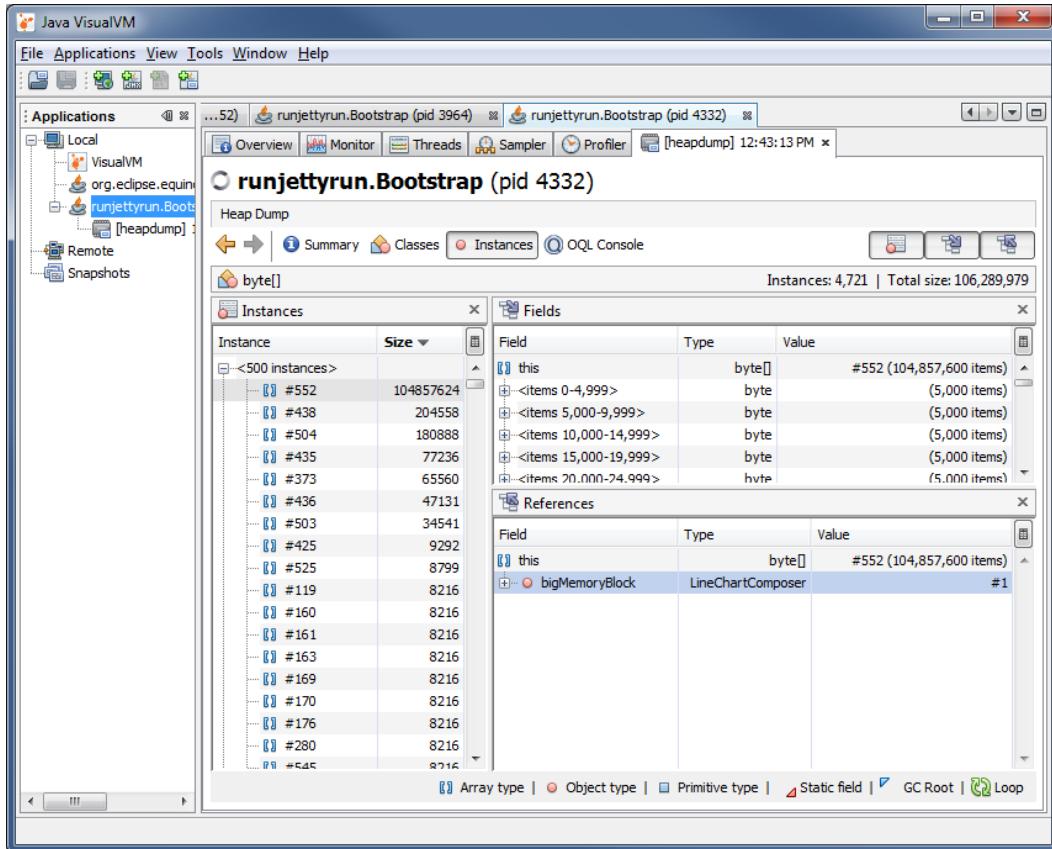
If we don't know where to look in the code we can use a Heap Dump to locate it.



I clicked the "find" button to find the 20 biggest objects on the heap. The results tell us the following.

- the 100MB is a big byte-array
- our code containing an equally big chunk of memory is the LineChartComposer
- the other classes look mostly session/desktop related

If it wasn't that obvious like here, one can always switch to the "Instances" view and check which objects are referring to this big array.



Also here we could trace the references up to the Desktop/Session objects.

In web applications it is a recurring problem that there are too many sessions consuming too much memory in ZK Desktops (containing the current component tree) for a user are stored in the session. Therefore one should check the following settings:

- ZK Session Cleaner^[10]

this listener is usually enabled in web.xml, so make sure it is not commented out, or removed.

- Session Configuration^[11]

check session timeout, either here or in web.xml

check max desktops per session, if you want to put a limit here

- Desktop Configuration^[12]

if desktops stay alive too long, check the desktop timeout

Usually desktops are destroyed when a page is closed, however in case a browser crashes, this mechanism will not work and the desktop will stay in the session, until it times out, consuming memory for the complete component tree of the orphaned desktop.

Also a good read: 10 Tips for using the Eclipse Memory Analyzer^[13]

Monitoring ZK

To check if you have a desktop/session leak, you can add the Statistic-listener to your application, and read its status in a simple monitoring page (keep in mind, that this page will consume an additional desktop).

Add a listener to zk.xml

```
<listener>
    <listener-class>org.zkoss.zk.ui.util.Statistic</listener-class>
</listener>
```

Example of a small monitoring page:

```
<zk>
    <label multiline="true">
        Active Desktops:
        ${desktop.webApp.configuration.monitor.activeDesktopCount}
        Active Sessions:
        ${desktop.webApp.configuration.monitor.activeSessionCount}

        Total Desktops:
        ${desktop.webApp.configuration.monitor.totalDesktopCount}
        Total Sessions:
        ${desktop.webApp.configuration.monitor.totalSessionCount}
    </label>
</zk>
```

Or you can access Statistics in Java code:

```
Statistic statistic = (Statistic)
WebApps.getCurrent().getConfiguration().getMonitor();
statistic.getActiveDesktopCount();
statistic.getActiveSessionCount();
```

related documentation

- http://books.zkoss.org/wiki/ZK_Configuration_Reference/zk.xml/The_listener_Element/The_org.zkoss.zk.ui.util.Monitor_interface
- <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Statistic.html>

If you see more Sessions than you expect, your session/desktop timeout might be too long, or too many desktops give you a hint that the desktop cleanup process is not functioning properly, also [reusing desktops^[14]] can help.

ZK Server Configuration

- check debug mode → zk config should **not** be enabled (disabled by default)
 - http://books.zkoss.org/wiki/ZK_Configuration_Reference/zk.xml/The_client-config_Element/The_debug-js_Element
- check caching config → should **not** be disabled (enabled by default)
 - http://books.zkoss.org/wiki/ZK_Configuration_Reference/zk.xml/The_Library_Properties/org.zkoss.web.classWebResource.cache
 - http://books.zkoss.org/wiki/ZK_Configuration_Reference/zk.xml/The_Library_Properties/org.zkoss.zk.WPD.cache

- http://books.zkoss.org/wiki/ZK_Configuration_Reference/zk.xml/The_Library_Properties/org.zkoss.zk.WCS.cache
- check compression settings → should **not** be disabled (enabled by default)
 - http://books.zkoss.org/wiki/ZK_Configuration_Reference/web.xml/ZK_AU_Engine#The_Initial_Parameters
 - http://books.zkoss.org/wiki/ZK_Configuration_Reference/web.xml/ZK_Initializer#The_Initial_Parameters
- consider/check render on demand settings
 - http://books.zkoss.org/wiki/ZK_Developer%27s_Reference/Performance_Tips/Client_Render_on_Demand
 - http://books.zkoss.org/wiki/ZK_Developer%27s_Reference/Performance_Tips/Listbox,_Grid_and_Tree_for_Huge_Data/Turn_on_Render_on_Demand
 - http://books.zkoss.org/wiki/ZK_Configuration_Reference/zk.xml/The_Library_Properties/org.zkoss.zul.client.rod
 - http://books.zkoss.org/wiki/ZK_Configuration_Reference/zk.xml/The_Library_Properties/org.zkoss.zul.grid.initRodSize
 - http://books.zkoss.org/wiki/ZK_Configuration_Reference/zk.xml/The_Library_Properties/org.zkoss.zul.listbox.initRodSize
 - [http://books.zkoss.org/wiki/ZK_Configuration_Reference/zk.xml/The_Library_Properties/org.zkoss.zul.tree.initRodSize_\(ZK_7\)](http://books.zkoss.org/wiki/ZK_Configuration_Reference/zk.xml/The_Library_Properties/org.zkoss.zul.tree.initRodSize_(ZK_7))
- check http://books.zkoss.org/wiki/ZK_Developer%27s_Reference/Performance_Tips

Network Issue

If something in your network infrastructure (routers, proxies, web servers...) is causing the performance issues, there is little you can do as a web application developer.

Here are some ideas to identify possible bottlenecks trying to reduce network complexity by:

- using ip addresses directly to circumvent DNS look-ups → speedup might indicate a problem with your DNS server
- avoiding proxies, routers, firewalls (e.g. access the application server from a browser on a remote desktop "closer" to the actual server)
- accessing the application server directly, instead of going through a webserver or load balancer
- disabling SSL and check difference

→ "Kindly" inform your network administrator about your observations and ask for help identifying, excluding, fixing these infrastructure problems.

Java Profiling Tool

If you find the performance bottleneck is at the server-side, you need to profile your java program to locate which method is most time-consuming with a java profiling tool.

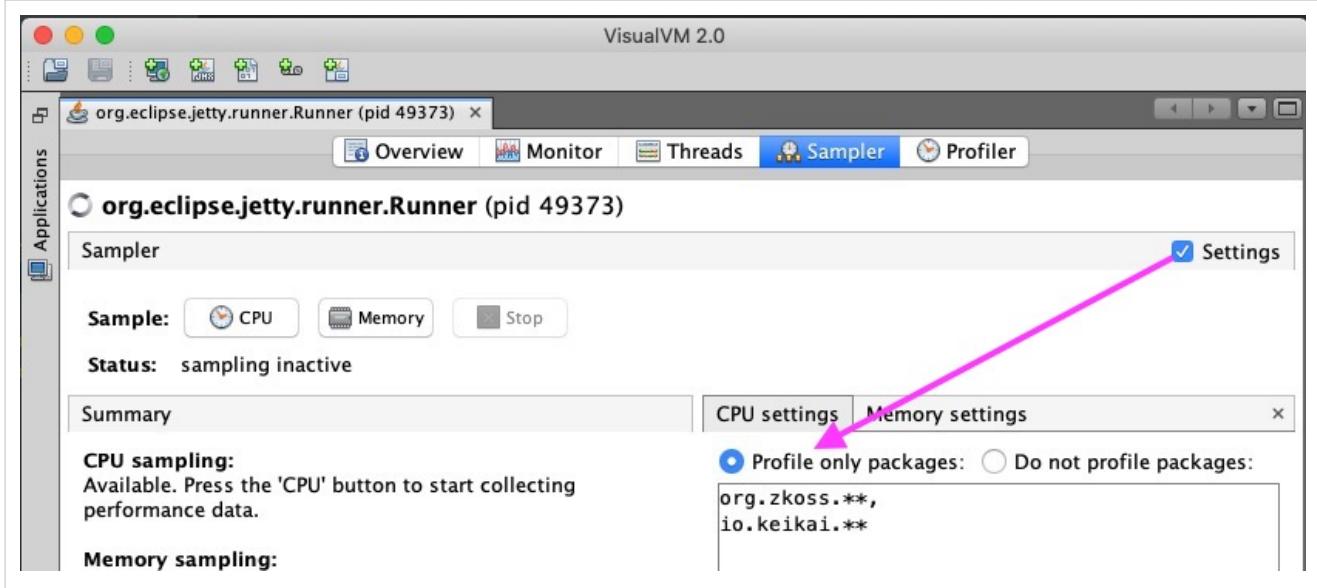
VisualVM^[15] is a java profiling tool bundled with JDK (or you can download from its website).

You can watch the following :

- Introduction to Java VisualVM^[16]
- Ninjas' guide to getting started with VisualVM^[17]

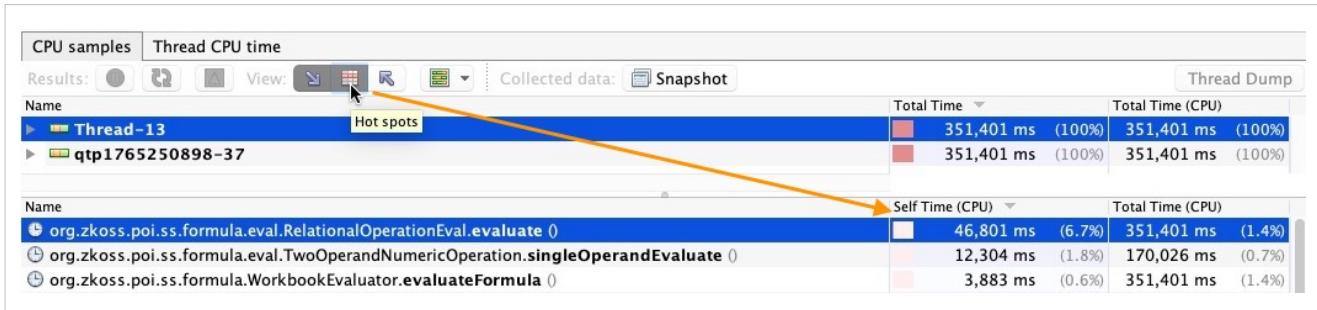
Limit Profile Packages

When using Sampler or Profiler, it's better to specify **Profile only packages** in the settings. Therefore, it will only show those classes you concern instead of irrelevant classes.



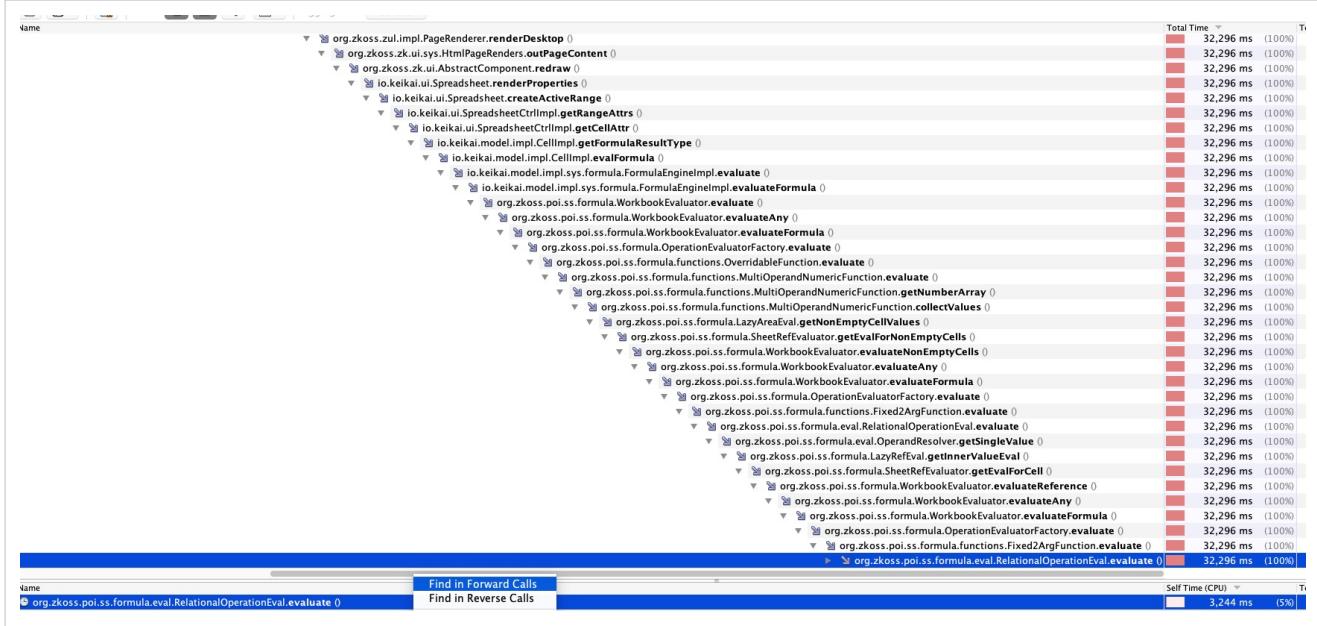
Locate Hot Spots

After you start the CPU sampler/profiler, you can click the "hotspot" to show you the most time-consuming method:



Show Calling Hierarchy

Right-click on a method, choose "Find in Forward Call", it will list the calling hierarchy to the selected method. Help you to locate source calling method.



References

- [1] <https://developer.chrome.com/devtools/docs/network#resource-network-timing>
- [2] http://books.zkoss.org/wiki/ZK_Developer%27s_Reference/Performance_Tips/Client_Render_on_Demand
- [3] http://books.zkoss.org/wiki/ZK_Developer%27s_Reference/Performance_Tips/Listbox,_Grid_and_Tree_for_Huge_Data/Use_Live_Data_and_Paging
- [4] <https://developers.google.com/web/tools/chrome-devtools/evaluate-performance/>
- [5] http://books.zkoss.org/wiki/ZK_Configuration_Reference/zk.xml/The_client-config_Element/The_debug-js_Element
- [6] <http://technet.microsoft.com/en-us/sysinternals/bb896653>
- [7] http://www.computerperformance.co.uk/HealthCheck/Disk_Health.htm#Disk%20Bottleneck%202
- [8] https://blogs.oracle.com/nbprofiler/entry/profiling_with_visualvm_part_1
- [9] https://blogs.oracle.com/nbprofiler/entry/profiling_with_visualvm_part_2
- [10] http://books.zkoss.org/wiki/ZK_Configuration_Reference/web.xml/ZK_Session_Cleaner
- [11] http://books.zkoss.org/wiki/ZK_Configuration_Reference/zk.xml/The_session-config_Element
- [12] http://books.zkoss.org/wiki/ZK_Configuration_Reference/zk.xml/The_desktop-config_Element
- [13] <http://eclipsesource.com/blogs/2013/01/21/10-tips-for-using-the-eclipse-memory-analyzer/>
- [14] http://books.zkoss.org/wiki/ZK_Developer's_Reference/Performance_Tips/Reuse/Desktops
- [15] <https://visualvm.github.io>
- [16] <https://www.youtube.com/watch?v=z8n7Bg7-A4I>
- [17] <https://engineering.talkdesk.com/ninjas-guide-to-getting-started-with-visualvm-f8bff061f7e7>

Accessibility

- Available for ZK:
-   

Overview - Achieve AA Compliance

According to W3C^[1], web accessibility means that websites, tools, and technologies are designed and developed so that people with disabilities can use them. This also benefits people without disabilities, for example, people with temporary disabilities (e.g. injured) or people who have situational limitations under particular circumstances (e.g. no mouse access).

The za11y (zk-accessibility) module enables you to create accessible WCAG 2.0^[2] **AA (Double-A)-compliant** applications. Key features include full keyboard support, assistive technologies support, and high-contrast themes.

To meet accessibility requirements, one key point is to provide semantic information about widgets, structures, and behaviors for assistive technologies. With the za11y module, ZK widgets will render such semantic information based on their purposes according to WAI-ARIA^[3]. However, depending on an application's context, a widget might play different roles in different pages and this has to be designed by application developers -- developers should fully understand the purposes, requirements, and expectations to deliver complete accessibility.

This chapter shares with you what the ZK framework has done in terms of accessibility in general, and how you can specify additional application-specific accessible information. For the details of each component, please refer to ZK Component Reference/Accessibility.

Enable Accessibility Support

ZK accessibility module is a separate, optional jar, you have to include it manually to enable it.

```
<dependency>
    <groupId>org.zkoss.zk</groupId>
    <artifactId>za11y</artifactId>
    <version>${zk.version}</version>
</dependency>
```

To ensure za11y.jar is running as expected at run-time, you can simply inspect a textbox. If you see ARIA attributes rendered in its DOM elements, that means the module works successfully. Please see Built-in Support.

We use axe DevTools 3.5.3 in ZK 9.5.0 to check the WCAG rules in za11y. We use Lighthouse 11.4.0 in ZK 10.0.0 to check the WCAG rules in za11y.

Built-in Support

To know what default ARIA attributes are rendered by ZK, please open the developer tool to inspect the widget.

textbox.zul

```
<textbox />
```

Visit the zul page and inspect the textbox with a browser developer tool, you will see some aria attributes rendered:

```
<input id="h5AP0" class="z-textbox" type="text"
       aria-disabled="false" ariareadonly="false">
```

Some components render special ARIA attributes, please refer to ZK Component Reference/Accessibility.

Specify ARIA Attributes

You can specify arbitrary ARIA attribute as needed on a component with namespace "client/attribute" [4] like:

```
<zk xmlns:ca="client/attribute">
    <div ca:aria-hidden="true"/>
    <textbox ca:aria-label="${field}" />
    <intbox ca:aria-labelledby="${price}" />
</zk>
```

Label with an Input Component

There are 3 ways to associate an input component with a label:

Enclose with a label

```
<zk xmlns:h="native" xmlns:ca="client/attribute">
    <h:label>Address
        <textbox/>
    </h:label>
```

Specify at aria-label

```
<zk xmlns:h="native" xmlns:ca="client/attribute">
    <custom-attributes field="Account:" />
    ${field}
    <textbox ca:aria-label="${field}" />
```

Specify at aria-labelledby

```
<zk xmlns:h="native" xmlns:ca="client/attribute">
    <label value="price" id="priceLabel"/>
    <textbox ca:aria-labelledby="${priceLabel.uuid}" />
```

Demo Application

This demo application shows how you can build a ZK-based application with za11y module support. It also demonstrates key accessible features such as high contrast themes, keyboard support, screen reader support, mouse/touch support, landmarks and so on.

Check demo online ^[5]

Check demo source at Github ^[6]

No. #	Status	No Items	Name	Zip Code	Total	Actions
1400	Open	3	Gregor Albert	89568	\$ 35.97	
1401	Open	4	Diethelm Ziegler	80606	\$ 53.96	
1402	Open	5	Diethelm Ziegler	80606	\$ 66.95	
1403	Open	5	Gero Höfler	25942	\$ 70.95	
1404	Open	3	Diethelm Ziegler	80606	\$ 43.97	
1405	Open	1	Nikola Stumpf	95409	\$ 14.99	

Screen Reader

During the development, we have tested the accessibility with the following screen readers:

JAW

- proprietary, Windows
- Toggle Virtual PC Cursor: Insert + Z or NumberPad +
- Guide ^[7]
- Issues ^[8]

NVDA

- Toggle browse/focus mode: Insert + Spacebar

VoiceOver

- Mac built-in, activate by Command + F5
- using VoiceOver to browse and navigate webpages ^[9]

Narrator ^[10]

- Windows 10 built-in
- start/stop narrator: windows + ctrl + enter
- switch scan mode: narrator (CapsLock) + space

References

- [1] <https://www.w3.org/WAI/fundamentals/accessibility-intro/>
- [2] <https://www.w3.org/TR/WCAG20>
- [3] <https://www.w3.org/TR/wai-aria/>
- [4] https://www.zkoss.org/wiki/ZUML_Reference/ZUML/Namespaces/Client_Attribute
- [5] <https://www.zkoss.org/za11y-demo>
- [6] <https://github.com/zkoss-demo/za11y-demo>
- [7] <https://www.freedomscientific.com/training/jaws/>
- [8] <https://github.com/FreedomScientific/VFO-standards-support/issues>
- [9] https://www.apple.com/voiceover/info/guide/_1134.html
- [10] <https://support.microsoft.com/en-us/windows/complete-guide-to-narrator-e4397a0d-ef4f-b386-d8ae-c172f109bdb1>

Keyboard Support

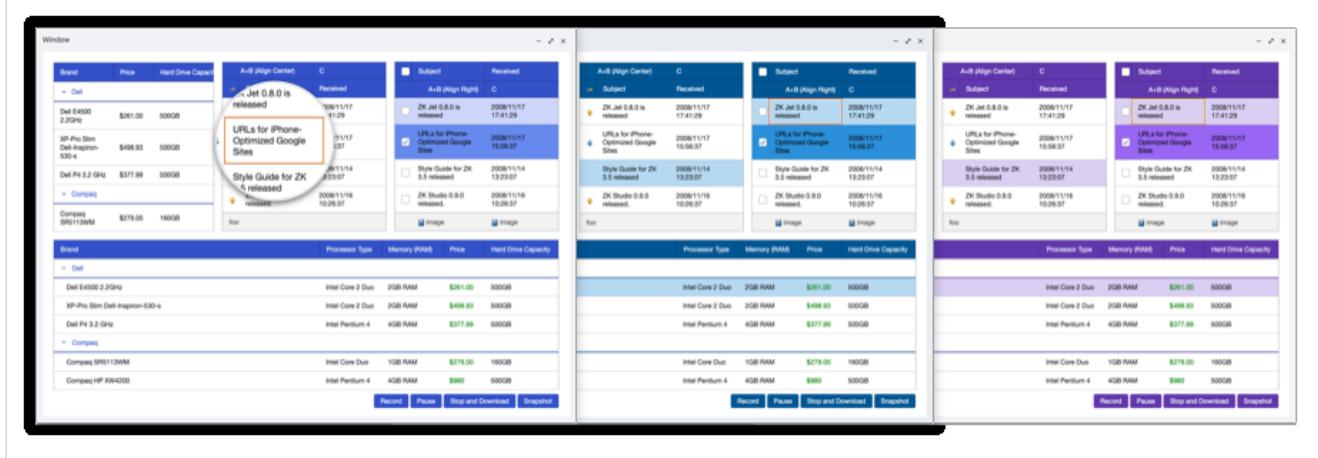
You can operate ZK components using only a keyboard, e.g. press TAB to move the focus and navigate. For keyboard support details of each component, please refer to ZK Component Reference/Accessibility.

To implement your own key handling, please see ZK Developer's Reference/UI Patterns/Keystroke Handling

High Contrast Theme

ZK provides 3 accessibility-ready high contrast themes ^[1] according to WCAG 1.4.3 Contrast (Minimum) ^[2]: *The visual presentation of text and images of text should have a contrast ratio of at least 4.5:1.* When we design these themes, we rely on Sketch Accessibility Assistant ^[3] to ensure each theme conforms to WCAG 2. After that, we have verified them using axe - Web Accessibility Testing ^[4].

To know more details, please refer to Accessibility-ready themes



References

- [1] https://www.zkoss.org/wiki/ZK_Developer's_Reference/Theming_and_Styling/ZK_Official_Themes#Accessibility-ready_themes
- [2] <https://www.w3.org/TR/WCAG20/>
- [3] <https://www.sketch.com/extensions/assistants/sketch-accessibility-assistant/>
- [4] <https://chrome.google.com/webstore/detail/axe-web-accessibility-tester/lhdoppojpmngadmnlindnejfepokejbdd>

Testing

ZK is a Java framework. Technically you could use any Java test tools you prefer. Here we describe the testing tips and ZTL (the official test tool based on Selenium).

ZK Jmeter plugin^[1]

The ZK Webdriver utility library^[2] is also available.

For information of particular test tools, please refer to small talks:

- Sahi: Making ZK Functional Tests With Sahi
- Selenium: How to Test ZK Application with Selenium and ZK Unit Testing

References

[1] <https://blog.zkoss.org/2013/08/06/zk-jmeter-plugin/>

[2] <https://github.com/zkoss/zk-webdriver>

Testing Tips

Here we introduce some tips when you use a browser testing tool to test ZK-based applications, e.g. JMeter^[1], TestCafe^[2], selenium.

Deal with Randomized UUID

By default, a desktop's ID and a component's UUID are randomized for preventing Cross-Site Request Forgery (CSRF) and allowing multiple desktops to coexist on the same web page (such as Portlet). A component's UUID is auto-generated by ZK and different from its ID. The UUID is used as a component DOM elements' id in a browser. However, it also means the DOM element's IDs will change from one test run to another.

If your test code runs at the server (such ZATS and JUnit), it is not an issue at all (since DOM elements are available at the client only). However, if your test tool runs in a browser, you have to locate an element with one of the following approaches:

1. Not to depend on a DOM element's ID. Rather, use a component's ID and/or component's parent-child-sibling relationship.
2. Implement IdGenerator^[3] to generate UUID in a predictable and repeatable way

Let me explain them in detail.

Approach 1: Locate by a component's ID

With Server+client architecture, ZK maintains an *identical* world at the client. If your test tool is able to access JavaScript at the client, your test code can depend on a component's ID and its widget's parent-child relationship as your application code depends on the component's ID and component's parent-child relationship. They are *identical*, except one is JavaScript and called Widget^[4], while the other is Java and called Component^[1].

This is a suggested approach since it is much easier to test an application at the same abstract level -- the component level, aka., the widget level (rather than the DOM level).

To retrieve widgets at the client, you can use one of the following JavaScript API:

- jq^[5]
- `_global_.Map` Widget.\$(zk.Object, _global_.Map)^[6]

jq^[5] allows your test code to access the components directly, so the test code could depend on a component's ID (Widget.id^[7]) and the widget tree (Widget.firstChild^[8], Widget.nextSibling^[9] and so on).

```
jq('@window[border="normal"]') //returns a list of window whose border  
is normal  
jq('$x'); //returns the widget whose component ID is x, <div id="x"/>  
jq('$x $y'); //returns the widget whose ID is y and it is in an ID  
space owned by x
```

With this approach, you still can verify the DOM structure if you want, since it can be retrieved from a widget's Widget.\$n()^[10].

ZTL^[11] is a typical example that takes this approach. For more information, please refer to the ZTL section.

Approach 2: Use ID Generator

If your testing tool running in a browser cannot access JavaScript, you can implement an ID generator to generate a desktop's ID and component's UUID in a predictable and repeatable manner.

Since ZK 7.0.0, ZK provides a static ID generator implementation for testing, to use StaticIdGenerator^[12], simply add it to zk.xml.

```
<system-config>  
  <id-generator-class>org.zkoss.zk.ui.impl.StaticIdGenerator</id-generator-class>  
</system-config>
```

To implement a custom ID generator, you have to do the following:

- Implement a Java class that implements IdGenerator^[3].
- Specify the Java class FQCN at id-generator-class element in zk.xml. For example,

```
<system-config>  
  <id-generator-class>my.IdGenerator</id-generator-class>  
</system-config>
```

Examples

- ComponentIdFirstGenerator^[13]. Unlike StaticIdGenerator^[14], component creation order doesn't affect id generation.
- StaticIdGeneratorExt^[15]. It generates desktop id in a predictable way.

Different Configuration for Different Environment

If you prefer to have a different configuration for the testing environment (such as specifying ID generator for testing), you could put the configuration in a separate file, say, WEB-INF/config/zk-testing.xml with the following content.

```
<zk>  
  <system-config>  
    <id-generator-class>my.IdGenerator</id-generator-class>  
  </system-config>  
</zk>
```

Then, you could specify -Dorg.zkoss.zk.config.path=/WEB-INF/config/zk-testing.xml as one of the arguments when starting the Web server.

Disabled UUID recycle

If you want to generate UUID with some conditions, you might also want to disable UUID recycling. (It will reuse all the UUIDs from removed components.) You could set the properties org.zkoss.zk.ui.uuidRecycle.disabled in zk.xml.

References

- [1] <https://jmeter.apache.org/>
- [2] <https://testcafe.io/>
- [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/IdGenerator.html#>
- [4] <http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#>
- [5] http://www.zkoss.org/javadoc/latest/jsdoc/_global_.jq.html#
- [6] [http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#\\${zk.Object,](http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#${zk.Object,)
- [7] <http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#id>
- [8] <http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#firstChild>
- [9] <http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#nextSibling>
- [10] [http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#\\${n\(\)](http://www.zkoss.org/javadoc/latest/jsdoc/zk/Widget.html#${n())
- [11] <http://code.google.com/p/zk-ztl/>
- [12] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/impl/StaticIdGenerator.html#>
- [13] <https://github.com/zkoss/zkbooks/blob/master/developersreference/developersreference/src/main/java/org/zkoss/reference/developer/testing/ComponentIdFirstGenerator.java>
- [14] <https://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/impl/StaticIdGenerator.html>
- [15] <https://github.com/zkoss/zkbooks/blob/master/developersreference/developersreference/src/main/java/org/zkoss/reference/developer/testing/StaticIdGeneratorExt.java>

ZATS

Overview

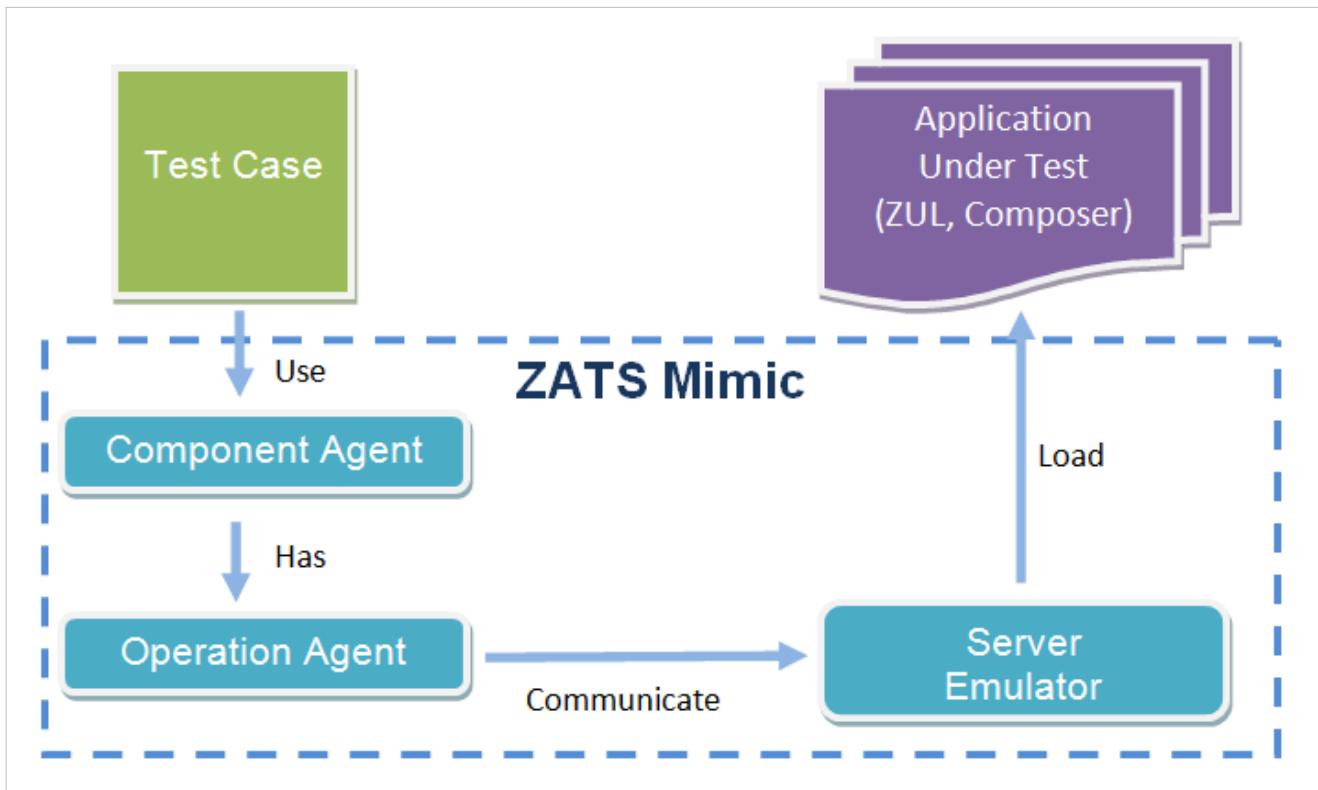
ZK Application Test Suite (ZATS) is a collection of tools which can help users test their ZK-based application. This suite has following modules:

ZATS Mimic, a unit-test library that can be used with any well-known unit test framework (e.g. JUnit and TestNG) to test your ZUL without an application server or a browser.

ZATS Mimic enables developers to test their composer **without an application server** and of course **without a browser** either. Through this library, testers can mimic user interactions with applications such as clicking or typing to verify composer's (controller layer) data and logic. All they have to do is to **write a regular unit test case** and use Mimic's utility class to interact with components on ZUL and then, run the test case.

No deploying to server, no rendering on browser, the unit test case can be executed in a very short period of time - this is very helpful for frequent unit testing during an agile development process.

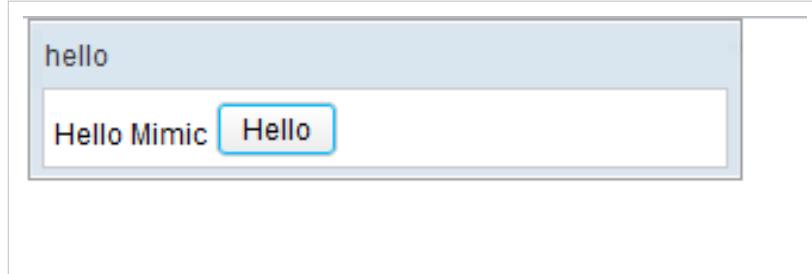
The concept is as follows:



Testers write test cases to simulate user actions such as clicking or typing with operation agents. Operation agent communicates with server emulator and triggers the composer's event handlers to change the component's status. Testers are able to check component's properties from the component agent to verify the result of user action. It might be a *label* changing its value or a *listbox* increasing by one item. **All behaviors that reflect on the component's properties can be verified.**

Test Case Sample

We have an application to test here. This application only has one label and one button with no other content at first. It has only one function: when a user clicks the button, the label shows "Hello Mimic" as shown in the image below. We are going to verify this behavior.



ZUL of our simple application

```
<zk>
    <window title="hello" border="normal" width="300px" apply="org.zkoss.zats.example.hello.HelloComposer">
        <label />
        <button label="Hello" />
    </window>
</zk>
```

Composer of our simple application

```
public class HelloComposer extends SelectorComposer {
    @Wire("label")
    Label label;

    @Listen("onClick = button")
    public void hello() {
        label.setValue("Hello Mimic");
    }
}
```

We write the test case with JUnit 4 annotation, please refer to JUnit 4 in 60 seconds [1].

The following test case will mimic a user clicking the button and verify whether or not the label's value is "Hello Mimic".

HelloTest.java"

```
//remove import for brevity
public class HelloTest {
    @BeforeClass
    public static void init() {
        Zats.init("./src/main/webapp");
    }

    @AfterClass
    public static void end() {
        Zats.end();
    }
}
```

```
}

@After
public void after() {
    Zats.cleanup();
}

@Test
public void test() {
    DesktopAgent desktop =
Zats.newClient().connect("/hello.zul");

    ComponentAgent button = desktop.query("button");
    ComponentAgent label = desktop.query("label");

    //button.as(ClickAgent.class).click();
    button.click();
    assertEquals("Hello Mimic",
label.as(Label.class).getValue());
}
}
```

- Before starting a test, we have to call `Zats.init()` and pass **root directory** where ZUL pages are stored as a parameter. Most of the time, it is located in your web application's content root folder. In our example, we use maven default project structure. This method initializes testing environment and starts the server emulator. (line 5)
- Of course, since we start the server emulator at `@BeforeClass`, we should stop it by `Zats.end()` at `@AfterClass`. (line 10)
- We should call `Zats.cleanup()` to clear desktop before opening another ZUL. (line 15)
- The first statement of a test case is to create a client and connect it to a ZUL page, like a browser visiting a ZUL. The `connect()` returns a `DesktopAgent` and we usually retrieve `ComponentAgent` from it to perform user operation. (line 20)
- Before we can mimic a user action to a component, we should **retrieve a ComponentAgent**. Empowered by selector syntax, `DesktopAgent .query()` is a powerful tool to retrieve it. As the ZUL contains only one button, we can query it by component name: `query ("button")` (line 22)
- As we do not have a browser screen to view, we cannot interact with a component by mouse's pointer. To mimic a user action, we have to convert `ComponentAgent` to one of the operation agents. The conversion method `as()` will check for available operation for the target `ComponentAgent`. For example, you cannot type something in a **Label**, If you try to convert it to an unsupported operation agent, you will get an exception. (line 25)
- For convenience, `ComponentAgent` provides shortcut methods for commonly used operations like `click()`; it automatically gets operation agent and calls it for you. (line 26)
- To verify test result, we can also use `ComponentAgent .as()` to convert it to a ZK component then get its property by getter methods. (line 27)

Version History

Version	Date	Content
6.0.0		overview

References

[1] <http://www.cavdar.net/2008/07/21/junit-4-in-60-seconds>

Upgrade Tips

This chapter describes how you can upgrade to a newer ZK version (ex. ZK 7 to ZK 9) or a different edition (ex. CE to PE/EE). It covers information regarding versioning, where to access release notes, where to download required jar files, recommended upgrade process and a list of major changes since ZK 7.

Version Upgrade

Versioning

ZK products follow semantic versioning ^[1], where the three digits refer to MAJOR.MINOR.PATCH respectively. In the case where a hotfix is required, a fourth digit is introduced. Except for major (first digit) upgrades, compatibility is generally maintained, or a fall-back mechanism is available.

Upgrade References

An introductory smalltalk is published for every MAJOR and MINOR release. For example New Features of ZK 9.5. The smalltalk highlights the most important changes as well as provides upgrade tips for upgrading to that specific version. View all New Features articles ^[2].

A detailed release note containing a list of fixed issues is available for every new release, including hotfixes. For example, ZK 9.0.0 release notes ^[3]. The full release history ^[4] can be found here.

An “upgrade notes” section will be included at the end of a release note when needed. The “upgrade notes” section points out API changes or changes that developers will most likely need to take actions accordingly.

You can follow the same naming conventions as the examples provided above to search for the upgrade references you are looking for.

Important/breaking changes

ZK Version	Description	Reference
ZK 5.0	1. A major architectural change was made in ZK 5, affecting theming, UI creation, and client-side rendering. Upgrading from ZK 2.x or ZK 3.x to ZK 5 requires more effort. 2. Event thread is now disabled by default. You can enable it in zk.xml if you need to use it.	Release Smalltalk Upgrading to ZK 5
ZK 6.0	1. Requires JDK 5 or higher	Release Smalltalk
ZK 6.5	1. Introduced Mobile/Tablet support, EE edition will have a different style and behavior on a mobile device. You can turn it off in zk.xml if that's not desired.	Release Smalltalk
ZK 7.0	1. Re-implement the theme with new client technologies including CSS3 and LESS, and the DOM structure of components changes a lot. To upgrade a custom theme for an older version, you will need to redo the style customization based on the new theme. Please refer to Upgrade From ZK 6.5. 2. Drop IE6 & IE7 support. 3. Deprecate several legacy packages and methods relating to session and event-thread.	Custom Style Upgrade Guide Release Notes [5]
ZK 8.0	1. A new form binding based on proxy is introduced. If you have been using MVVM org.zkoss.bind.SimpleForm, org.zkoss.bind.impl.FormImpl, and org.zkoss.bind.impl.FormExt you have to migrate it to the new proxy binding, or, upgrade to ZK 9.5 which supports the legacy simple form binding.	ZK 8 new form binding [6] Release Notes [7]
ZK 8.5	1. A new and default theme Iceblue is introduced. The new theme has larger margin and padding from the previous themes. If you are not ready to use the new default theme, you can switch back to the previous themes, or, use ZK 8.6's compact theme. 2. Drop IE8 support.	Release Smalltalk Release Notes [8]
ZK 9.0	1. ZK 9 requires JDK 8 or later versions. 2. ZK Flex uses CSS 3 Flex instead. You can disable css flex and fall back to the previous implementation if needed. 3. Packaging change: ZK Data binding 1 is moved to "zkplus-legacy"; ZK DSP Library is moved to a new module "zweb-dsp".	Release Smalltalk Release Notes [3]
ZK 9.1	1. The underlying jQuery version has been upgraded from jQuery 1.12 to jQuery 3.5. If you have custom javascript code that relies on jQuery, you may need to update accordingly.	Release Notes [9]
ZK 9.5	1. Optional libraries(slf4j-jdk14 and closure-compiler-unshaded) are no longer provided by default. You can plugin your preferred libraries if needed.	Release Smalltalk
ZK 9.6	1. The default desktop ID generator was replaced by a more secured one. If your tests depend on the previous generator, you can set it in system-config. 2. Deprecated the method of isEditionValid() and encodeWithZK() of org.zkoss.zk.fn.ZkFns and core.dsp.tld. 3. Since ZK 9.6, a JakartaEE-compatible package is released along with the current JavaEE-compatible package. For Jakarta EE 9 support, please use 9.6.0-jakarta version instead. 4. Since 9.6.0, the transitive dependency of jasperreports was removed in zkex. To use the jasperreport component, please add it manually.	Release Smalltalk
ZK 10.0	1. ZK 10 requires JDK 11 or later versions. 2. Deprecate old themes: Breeze, Sapphire, Silvertail, and Atlantic 3. ZK 10 works with modern browsers; IE11 and lower IE versions are no longer supported. 4. Enable InaccessibleWidgetBlockService by default in the EE version. Note that it will block Echo Event if the component is disabled or invisible. You can configure it using the library-property org.zkoss.zkmax.au.IWBS.disable. 5. ZK's corresponding features now require Java EE 7 API level, including Servlet 3.1, Bean Validation 1.1, EL 3.0, and JSP 2.3. 6. Remove all deprecated Java APIs and the legacy module "zkplus-legacy". Reference the release smalltalk and the linter tool for upgrade guidance. 7. ZK frontend is now typescript, if you have JS-based custom client-side, you may need to double-check.	Release Smalltalk

Check all security fixes in the tracker^[10]

Upgrade Process

Upgrading from ZK 5 (or later) to the latest version should be manageable. The steps are:

Specify the version you wish to upgrade to in your maven POM file, or, download corresponding binary files manually and put them into your project. It is important to make sure all ZK jar files are in the same version.

Then, fix any compilation errors and configure the required fallback settings by referencing the Upgrade References.

Checklist

To estimate the upgrading effort, we suggest a checklist:

1. Check Custom Components

check if there is any ZK_Component_Development_Essentials/Creating_the_Configuration_Files/The_language-addon that might declare custom components. Since you might need to review it after the upgrade.

2. Check configuration description (zk.xml)

Check each property in zk.xml upon ZK Configuration Reference to see if there is any change in the new version.

3. Check Custom Appearance (CSS)

If you customize a component's appearance with CSS, you might need to review it for the new version.

4. Check Programs Using ZK JavaScript API

If you customize a component with JavaScript, you might need to review it for the new version.

5. Check Bug Patches

Check Bug Tracker^[11] or release note^[4] for the existing bug patch. If that bug is fixed in the new version, you can remove the patch.

6. Check for Java Compiler Error

References

- [1] <https://semver.org/>
- [2] https://www.zkoss.org/wiki/Category>New_Features
- [3] <https://www.zkoss.org/product/zk/releasenote/9.0.0>
- [4] <https://www.zkoss.org/product/zk/releasenote/>
- [5] <https://www.zkoss.org/product/zk/releasenote/7.0.0>
- [6] <https://blog.zkoss.org/2015/02/03/zk8-new-form-binding-approach/>
- [7] <https://www.zkoss.org/product/zk/releasenote/8.0.0>
- [8] <https://www.zkoss.org/product/zk/releasenote/8.5.0>
- [9] <https://www.zkoss.org/product/zk/releasenote/9.1.0>
- [10] <https://tracker.zkoss.org/issues/?jql=labels%20%3D%20security>
- [11] <https://tracker.zkoss.org/>

Edition Upgrade

Feature comparison

ZK Framework is offered in 3 editions, CE, PE, and EE. CE is the basic version, and EE comes with the most complete features. To see what features are included in each edition, visit the Feature Page ^[1].

Note that some of the add-ons like ZK Charts, Pivottable, and Keikai Spreadsheet are not part of ZK CE, PE, or EE. They can be used with ZK CE, PE, and EE, but separate licenses are required to use these add-ons.

Upgrade the Library

To upgrade from CE to ZK PE or EE (or PE/EE Evaluation Copy), download the corresponding PE/EE (or Evaluation) binary files from the Download Page ^[2], or specify the evaluation repository and include required PE/EE dependencies if you use maven. Maven instructions here.

Download instructions for paying customers were given at the time of the purchase. If you are not sure, contact us ^[3] with your License Certificate Number and we will help.

Evaluation vs. Official version

We offer 60-day free Evaluation copies for users to try out ZK PE and EE. The Evaluation copy basically has the same technical capabilities as the commercially licensed version, except for a few differences:

- 1) With the evaluation copy, it is expected to see the following warning in your server console.

SEVERE: This is an evaluation copy of ZK EE and will terminate after sixty days from the first date of installation. Should you require an open source license or commercial license for ZK EE please contact us at info@zkoss.org for more information. Alternatively you can download ZK CE which is licensed under the LGPL.

Depending on the product you use, the message may be slightly different. This message is to remind you that you are currently with an Evaluation copy which is for evaluation only and cannot be used in a production server.

- 2) With the evaluation copy, your application will show an uptime warning when it reaches a configured time (normally between 8 and 72 hours) and may block your ZK pages. If you are still within your 60-day trial period, you can restart your webserver to continue your evaluation.

If you develop your application based on the evaluation copy, when you are ready to move to the official (paid) version, you do not need to change your application code. All you need to do is to switch the library from the evaluation repo/jars to the paid repo/jars, following the download instructions we provide you at the time of your purchase.

Include required modules

In addition to the core ZK features, PE/EE users are given access to use certain add-ons and tools according to the package they are on.

For example, the `zuti` module is an EE-only feature and is required if you wish to use shadow elements. The `zally` package is also EE-only and is required if you wish to have Web accessibility support. Some features are located in the same repository, but others are in a separate repository. For example, a EE user can use ZATS Mimic, ZK Spring, ThemePack themes and ZK Calendar - they are not packaged in the core ZK. You will need to download them from their download pages, or, include the dependency in your maven POM file.

If you wish to use a specific feature, reference the document page of that feature, or the content (jar) of binary distributions to see if it is already in the core package, or if you need to specify a different jar file or dependency.

Use PE/EE specific components and features

After upgrading to PE/EE, you can access more components, features, and Java classes. Please read the documentation of each feature to learn how to set it up.

Features enabled by default

After upgrading to PE/EE, some of the features are enabled by default. For example, the client render on demand feature will be enabled by default to improve the client-side performance. Also, with EE, the Tablet UI will be enabled by default, when a tablet or mobile device connects to your application the tablet-supported components will switch themselves to the tablet theme. You can manually disable them in zk.xml if you are not ready.

Change of default behavior

Some of the default behavior changes after you upgraded the ZK edition. For example in the case of server push, with CE, it uses client polling. Once you upgrade to PE, the default push becomes comet push, and with EE it uses servlet 3 comet. Learn more here.

Require configuration

Some PE/EE features are now available for you to use, but you will have to enable them by yourself. For example if you have a big grid or listbox, you can turn on render on demand to boost its performance.

Please reference the documentation of each feature to see if any configuration is required and contact us ^[3] if you have any questions.

References

- [1] <https://www.zkoss.org/whyzk/Features>
 - [2] <https://www.zkoss.org/download/zk?ee>
 - [3] <https://www.zkoss.org/support/about/contact>
-

Customization

Here describes how to customize ZK, such as initializing components with different properties, loading ZUML document from database and so on.

Packing Code

There are two ways to pack the customization code: part of the Web application, or an independent JAR file. Packing as part of the Web application is straightforward. All you have to do is to specify the customization in WEB-INF/zk.xml as described in ZK Configuration Reference.

In many cases, it is better to pack the customization code as an independent JAR file, such that it can be managed separately and reused in multiple Web applications.

Where to Configure a JAR File

The configuration of a JAR file can be placed in a file called config.xml, and it must be placed under /metainfo/zk. If the JAR file also provides the component definitions, you have to prepare another file called lang-addon.xml under the same directory^[1].

The content of /metainfo/zk/config.xml is similar to WEB-INF/zk.xml, except only a subset of configurations is allowed. Here is a sample (zkex.jar's config.xml)^[2]:

```
<config>
    <config-name>zkex</config-name><!-- used to resolve dependency -->
    <depends>zk</depends>

    <version>
        <version-class>org.zkoss.zkex.Version</version-class>
        <version-uid>5.0.6</version-uid>
        <zk-version>5.0.0</zk-version><!-- or later -->
    </version>

    <listener>
        <listener-class>org.zkoss.zkex.init.WebAppInit</listener-class>
    </listener>

    <library-property>
        <name>org.zkoss.zul.chart.engine.class</name>
        <value>org.zkoss.zkex.zul.impl.JFreeChartEngine</value>
    </library-property>
    <library-property>
        <name>org.zkoss.zul.captcha.engine.class</name>
        <value>org.zkoss.zkex.zul.impl.JHLabsCaptchaEngine</value>
    </library-property>
</config>
```

[1] For more information, please refer to ZK Client-side Reference: Language Definition.

[2] For more information, please refer to ZK Configuration Reference: JAR File's config.xml.

How to Initialize a JAR File

Sometimes you have to initialize a JAR file. It can be done by implementing WebAppInit (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/WebAppInit.html#>), and then specifying it as a listener in /metainfo/zk/config.xml. For example,

```
public class MyJARInit implements WebAppInit {  
    public void init(WebApp wapp) throws Exception {  
        //do whatever init you need  
    }  
}
```

Notice that many configurations can be done by accessing Configuration (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Configuration.html#>) directly. If you want to access it in WebAppInit.init(org.zkoss.zk.ui.WebApp) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/WebAppInit.html#init\(org.zkoss.zk.ui.WebApp\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/WebAppInit.html#init(org.zkoss.zk.ui.WebApp))), invoke WebApp.getConfiguration() ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/WebApp.html#getConfiguration\(\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/WebApp.html#getConfiguration())) as follows.

```
public void init(WebApp wapp) throws Exception {  
    Configuration config = wapp.getConfiguration();  
    ...
```

Component Properties

With component definitions, we could specify the initial values for the properties, attributes and annotations of a component.

Properties

Depending on the requirement, you could change the initial value of a property for a particular ZUML document or for the whole application.

Notice that the initial values are applicable only to the component instantiated by ZK Loaders. It has no effect if you instantiate it in pure Java (unless you invoke Component.applyProperties()^[4] after instantiating a component).

Page-wide Initialization

Suppose we want to assign `normal` to the border property (`Window.setBorder(java.lang.String)`^[1]) of all windows in a ZUML document, then we could use the component directive as follows.

```
<?component name="window" extends="window" border="normal"?>
<window title="Border"/>
```

Application-wide Initialization

If you prefer to have the same initial value for all ZUML documents, you could specify it in a language addon. For example, we could prepare a file called `WEB-INF/lang-addon.xml` with the following content:

```
<language-addon>
  <addon-name>myapp</addon-name>
  <language-name>xul/html</language-name>
  <component>
    <component-name>window</component-name>
    <extends>window</extends>
    <property>
      <property-name>border</property-name>
      <property-value>normal</property-value>
    </property>
  </component>
</language-addon>
```

Then, we could specify this file by adding the following content to `WEB-INF/zk.xml`:

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

For more information, please refer to ZK Configuration Reference.

Molds

A mold is yet another property (`Component.setMold(java.lang.String)`^[1]), so you could change the initial value as described in the previous section. However, since it is common to change the value, we allow developers to specify the mold for a given component in a library property. As shown, the library is named as `ClassName.mold`. For example, if you would like to specify `trendy` as the initial mold of a button, then you could add the following to `WEB-INF/zk.xml`:

```
<library-property>
  <name>org.zkoss.zul.Button.mold</name>
  <value>trendy</value>
</library-property>
```

Attributes

Like properties, you could assign an attribute's initial value for a given component in the whole application (like calling Component.setAttribute()^[2]).

Notice that the initial values are applicable only to the component instantiated by ZK Loaders. It has no effect if you instantiate it in pure Java (unless you invoke Component.applyProperties()^[4] after instantiating a component).

Page-wide Initialization

Unlike the initial value of a property, there is no way to specify the initial value of a custom attribute in a ZUML document.

Application-wide Initialization

Similar to customizing the initial value of a property, you could specify the following in a language addon to assign an initial value of a attribute to a component.

```
<language-addon>
    <addon-name>myapp</addon-name>
    <language-name>xul/html</language-name>
    <component>
        <component-name>panel</component-name>
        <extends>panel</extends>
        <custom-attribute>
            <attribute-name>any.attribute.name</attribute-name>
            <attribute-value>any.value</attribute-value>
        </custom-attribute>
    </component>
</language-addon>
```

References

- [1] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Window.html#setBorder\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zul/Window.html#setBorder(java.lang.String))
- [2] [https://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/ext/Scope.html#setAttribute\(java.lang.String,%20java.lang.Object,%20boolean\)](https://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/ext/Scope.html#setAttribute(java.lang.String,%20java.lang.Object,%20boolean))

UI Factory

UiFactory^[1] is used to instantiate all UI objects, such as session, desktop, and components, and to load ZUML documents. You could customize it to provide the functionality you want.

For example, `SerializableUiFactory`^[1] is the factory used to instantiate sessions that are serializable^[1], while `SimpleUiFactory`^[2], the default factory, instantiates non-serializable sessions.

Here are a list of customization you could do with UI Factory:

- Load a ZUML document from, say, a database
 - It can be done by overriding `java.lang.String`) `UiFactory.getPageDefinition(org.zkoss.zk.ui.sys.RequestInfo, java.lang.String)` [3]
 - Instantiate a component by using a different implementation
 - It can be done by overriding `org.zkoss.zk.ui.Component`, `org.zkoss.zk.ui.metainfo.ComponentInfo`) `UiFactory.newComponent(org.zkoss.zk.ui.Page, org.zkoss.zk.ui.Component, org.zkoss.zk.ui.metainfo.ComponentInfo)` [4] and `org.zkoss.zk.ui.Component`, `org.zkoss.zk.ui.metainfo.ComponentInfo, java.lang.String`) `UiFactory.newComponent(org.zkoss.zk.ui.Page, org.zkoss.zk.ui.Component, org.zkoss.zk.ui.metainfo.ComponentInfo, java.lang.String)` [4].
 - Instantiate a desktop by using a different implementation
 - It can be done by overriding `java.lang.String, java.lang.String`) `UiFactory.newDesktop(org.zkoss.zk.ui.sys.RequestInfo, java.lang.String, java.lang.String)` [5]
 - Instantiate a page by using a different implementation
 - It can be done by overriding `org.zkoss.zk.ui.metainfo.PageDefinition, java.lang.String`) `UiFactory.newPage(org.zkoss.zk.ui.sys.RequestInfo, org.zkoss.zk.ui.metainfo.PageDefinition, java.lang.String)` [6] and/or `org.zkoss.zk.ui.Richlet, java.lang.String`) `UiFactory.newPage(org.zkoss.zk.ui.sys.RequestInfo, org.zkoss.zk.ui.Richlet, java.lang.String)` [6]

Notice that it is suggested to extend from either `SerializableUiFactory`^[1] or `SimpleUiFactory`^[2], rather than to implement `UiFactory`^[1] from scratch.

[1] Then, the application is able to run in a clustering environment. For more information, please refer to the Clustering section

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

[3] [RequestInfo,](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/UiFactory.html#getPageDefinition(org.zkoss.zk.ui.sys.RequestInfo,)

[5] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/UiFactory.html#newDesktop\(org.zkoss.zk.ui.sys.RequestInfo,](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/UiFactory.html#newDesktop(org.zkoss.zk.ui.sys.RequestInfo,)

[6] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/UiFactory.html#newPage\(org.zkoss.zk.ui.sys.RequestInfo,](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/UiFactory.html#newPage(org.zkoss.zk.ui.sys.RequestInfo,)

Load ZUML from Database

The default implementation of `java.lang.String`) ([`AbstractUiFactory.getPageDefinition\(org.zkoss.zk.ui.sys.RequestInfo, java.lang.String\)`](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/impl/AbstractUiFactory.html#getPageDefinition(org.zkoss.zk.ui.sys.RequestInfo, java.lang.String))) loads the ZUML document from the Web application's resources (i.e., the files found in a Web application). If you prefer to load from other sources, such as a database, you could override it.

The pseudo code will look like the following:

```
public class MyUiFactory extends SimpleUiFactory {  
    @Override  
    public PageDefinition getPageDefinition(RequestInfo ri, String  
path) {
```

```

        PageDefinition pgdef = getFromCache(path); //your cache
implementation
        if (pgdef == null) {
            String content = loadFromDatabase(path); //your resource
loading
            pgdef = getPageDefinition(ri, content, "zul"); //delegate
to SimpleUiFactory
            setCache(path, pgdef); //cache the result
        }
        return pgdef;
    }
}

```

where we assume you implemented `loadFromDatabase` to load the ZUML document from a database. In addition, you have to implement `getFromCache` and `setCache` to cache the result in order to improve the performance of retrieving the document from the database.

On the other hand, the parsing of the ZUML document can be done easily by calling `java.lang.String, java.lang.String) AbstractUiFactory.getPageDefinitionDirectly(org.zkoss.zk.ui.sys.RequestInfo, java.lang.String, java.lang.String)` ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/impl/AbstractUiFactory.html#getPageDefinitionDirectly\(org.zkoss.zk.ui.sys.RequestInfo,.\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/impl/AbstractUiFactory.html#getPageDefinitionDirectly(org.zkoss.zk.ui.sys.RequestInfo,.))).

Life Cycle Listener

You could have some custom initialization and cleanup when an application, a session, a desktop or an execution is instantiated or about to be destroyed.

There are two steps:

1. Implements the corresponding interface. For example, `WebAppInit`^[5] for application's initialization
2. Register it in `WEB-INF/zk.xml`, or in Java.

Interfaces

Task	Interface
Application Init	<code>WebAppInit</code> ^[5]
Application Cleanup	<code>WebAppCleanup</code> ^[1]
Session Init	<code>SessionInit</code> ^[2]
Session Cleanup	<code>SessionCleanup</code> ^[3]
Desktop Init	<code>DesktopInit</code> ^[4]
Desktop Cleanup	<code>DesktopCleanup</code> ^[3]
Execution Init	<code>ExecutionInit</code> ^[5]
Execution Cleanup	<code>ExecutionCleanup</code> ^[4]

Notice that ZK will instantiate an object from the class you registered for each callback. For example, an object is instantiated to invoke `java.lang.Object` `DesktopInit.init(org.zkoss.zk.ui.Desktop, java.lang.Object)`^[6], and another object instantiated to invoke `DesktopCleanup.cleanup(org.zkoss.zk.ui.Desktop)`^[7], even if you register a class that implements both `DesktopInit`^[4] and `DesktopCleanup`^[3].

If you have something that is initialized in the init callback and have to clean it up in the cleanup callback, you cannot store it as a data member. Rather, you have to maintain it by yourself, such as storing it in the desktop's attributes (`java.lang.Object` `Desktop.setAttribute(java.lang.String, java.lang.Object)`^[8]), session's attributes or application's attributes.

Registration

The registration in `WEB-INF/zk.xml` is the same, no matter what interface you implement:

```
<listener>
    <listener-class>my.MyImplementation</listener-class>
</listener>
```

The registration in Java is done by `Configuration.addListener(java.lang.Class)`^[9].

```
webapp.getConfiguration().addListener(my.MyImplementation.class);
```

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/WebAppCleanup.html#>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/SessionInit.html#>
- [3] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/SessionCleanup.html#>
- [4] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/DesktopInit.html#>
- [5] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/ExecutionInit.html#>
- [6] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/DesktopInit.html#init\(org.zkoss.zk.ui.Desktop,](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/DesktopInit.html#init(org.zkoss.zk.ui.Desktop,)
- [7] [>http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/DesktopCleanup.html#cleanup\(org.zkoss.zk.ui.Desktop\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/DesktopCleanup.html#cleanup(org.zkoss.zk.ui.Desktop))
- [8] [>http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Desktop.html#setAttribute\(java.lang.String, java.lang.Object\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Desktop.html#setAttribute(java.lang.String, java.lang.Object))
- [9] [>http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Configuration.html#addListener\(java.lang.Class\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Configuration.html#addListener(java.lang.Class))

AU Services

An AU service (AuService^[2]) is a plugin used to intercept the AU requests (AuRequest^[1]) sent from the client.

By plugging in an AU service, you could

- Ignore some AU requests (such as hostile requests)
- Change the default way of handling an AU request
- Handle application-specific AU requests

To plug an AU service to a desktop, you could invoke Desktop.addListener(java.lang.Object)^[2]. You could plug as many AU services as you want. Once plugged, all AU requests will go through the AU services (unless it was ignored by other AU service).

If you want to plug a particular component, you could invoke Component.setAuService(org.zkoss.zk.au.AuService)^[3]. Unlike desktops, a component can have at most one AU service.

If you want to plug an AU service, you could implement DesktopInit^[4] and register it in zk.xml as described in Life Cycle Listener.

```
public class MyDesktopInit implements DesktopInit {  
    public void init(Desktop desktop, Object request) {  
        desktop.addListener(new MyAuService()); //assume you have an AU  
        service called MyAuService  
    }  
}
```

References

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

[2] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/Desktop.html#addListener\(java.lang.Object\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/sys/Desktop.html#addListener(java.lang.Object))

[3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#setAuService\(org.zkoss.zk.au.AuService\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/Component.html#setAuService(org.zkoss.zk.au.AuService))

AU Extensions

An AU extension (AuExtension^[1]) is a small program that can be plugged into ZK Update Engine (DHtmlUpdateServlet^[2]) and extend its functionality. Actually our file upload and multimedia viewing are implemented as an AU extension that you can replace with your implementation.

An AU extension is associated with a name starting with slash, such as "/upload". Then each time a request targeting /zkau/upload will be forwarded to this extension for service.

To register an AU extension, you could specify the name and the class name as the initial parameter of the declaration of ZK Update Engine in WEB-INF/web.xml. For more information, please refer to ZK Configuration Reference.

If you want to register it in Java, you could use org.zkoss.zk.au.http.AuExtension) DHmlUpdateServlet.addAuExtension(java.lang.String, org.zkoss.zk.au.http.AuExtension)^[3] instead.

References

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

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

[3] [http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/au/http/DHmlUpdateServlet.html#addAuExtension\(java.lang.String,org.zkoss.zk.au.http.AuExtension\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/au/http/DHmlUpdateServlet.html#addAuExtension(java.lang.String,org.zkoss.zk.au.http.AuExtension))

How to Build ZK Source Code

 **Note:** The content of this page has been deprecated/removed in the latest version.

SVN Repository

Depending on the branch you want, you could check out the source codes from the following paths.

Version	URL	Description
5.0.* (current)	[1]	The 5.0 branch. It is the working repository for the most up-to-date source codes for current ZK 5
6.0.* (upcoming)	[2]	The 6.0 branch. It is the working repository for the most up-to-date source codes for upcoming ZK 6
3.6.* (maintaining)	[3]	The 3.6 branch. It is the working repository for the most up-to-date source codes for maintaining ZK 3.6. Though it is named <i>trunk</i> , it is used only for the 3.6 branch now.
3.0.* (maintaining)	[4]	The 3.0 branch. It is the working repository for the most up-to-date source codes for maintaining ZK 3.0
2.4.* (maintaining)	[5]	The 2.4 branch. It is the working repository for the most up-to-date source codes for maintaining ZK 2.4
Releases (frozen)	https://zk1.svn.sourceforge.net/svnroot/zk1/releases/x.y.z	The releases. We won't change the code in this repository. The URL depends on the version you want to check out. For a complete list, please visit [6]

Maven Build

Since ZK 5.0.5 release, we put an Eclipse project setting into each submodule folder, such as `zk`, `zul`, `zkdemo`, and so on. After that time, you can check out the source code from the SVN path above as an Eclipse Maven project to develop/build it. Or you can use Maven command line to build the ZK Jar files.

For example,

```
$ svn checkout  
https://zk1.svn.sourceforge.net/svnroot/zk1/releases/5.0.5/zul zul  
$ cd zul  
$ mvn clean package
```

See Also

- [ZK Installation Guide/Maven Setup](#)

References

- [1] <https://zk1.svn.sourceforge.net/svnroot/zk1/branches/5.0/>
- [2] <https://zk1.svn.sourceforge.net/svnroot/zk1/branches/6.0/>
- [3] <https://zk1.svn.sourceforge.net/svnroot/zk1/trunk/>
- [4] <https://zk1.svn.sourceforge.net/svnroot/zk1/branches/3.0/>
- [5] <https://zk1.svn.sourceforge.net/svnroot/zk1/branches/2.4/>
- [6] <http://zk1.svn.sourceforge.net/viewvc/zk1/releases/>.

Handle AU Request Resend

When an AU request fails, the default *Client Engine* implementation will retry 3 times to resend it and ask for a confirming dialog to user.

We provide a way for developer to customize the error handling.

For example,

```
zAu.ajaxErrorHandler = function (req, status, statusText, ajaxReqTries)  
{  
    if (ajaxReqTries == null)  
        ajaxReqTries = 3; // retry 3 times  
  
    // reset the resendTimeout, for more detail, please refer to  
    //  
http://books.zkoss.org/wiki/ZK\_Configuration\_Reference/zk.xml/The\_client-config\_Element/The\_auto-resend-timeout\_Element  
  
    zk.resendTimeout = 2000; //wait 2 seconds to resend.  
  
    if (!zAu.confirmRetry("FAILED_TO_RESPONSE", status+(statusText?":  
"+statusText:"")))  
        return 0; // no retry;  
    return ajaxReqTries;  
}
```

For more detail on the arguments, please take a look at int, _global_.String, int) zAu.ajaxErrorHandler(java.lang.Object, int, _global_.String, int)^[1]

Version History

Version	Date	Content
6.5.2	February, 2013	AU response error can be handle by user ^[2]

References

[1] [http://www.zkoss.org/javadoc/latest/jsdoc/_global/_zAu.html#ajaxErrorHandler\(java.lang.Object,](http://www.zkoss.org/javadoc/latest/jsdoc/_global/_zAu.html#ajaxErrorHandler(java.lang.Object,)

[2] <http://tracker.zkoss.org/browse/ZK-1616>

Supporting Utilities

In this section we will discuss the utilities that ZK is built on. You don't need them to develop ZK applications, but you might find them useful if they are applicable. Here we provide the basic information of them. Interested readers might refer to Javadoc^[1] for detailed API.

References

[1] <http://www.zkoss.org/javadoc/latest/zk/>

Logger

In this section we describe how to configure the logging of ZK internal functions. You can generally ignore it, unless you'd like to know how ZK operates internally.

Notice that, if you are using Google App Engine, you can *not* configure the logging as described in this chapter. For more information, please refer to Setting up Google App Engine.

How to Configure Logging

ZK uses SLF4J^[1] as its internal logging system, and developers can follow the SLF4J document^[2] to use the logging in ZK.

By default, ZK maven setting will bundle the **slf4j-jdk14** implementation as its default logging.^[3] If developers want to change that implementation for **Log4j**, **Simple**, or **Logback**, they have to exclude the dependency first like:

```
<dependency>
    <groupId>org.zkoss.common</groupId>
    <artifactId>zcommon</artifactId>
    <version>${zk.version}</version>
    <exclusions>
        <exclusion>
            <groupId>org.slf4j</groupId>
```

```
<artifactId>slf4j-jdk14</artifactId>
</exclusion>
</exclusions>
</dependency>
```

ZK maven dropped the transitive dependency of **slf4j-jdk14**. If a developer decides to enable logging, include either **Log4j**, **Simple**, or **Logback** implementation in their maven pom.xml file, e.g.

```
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>${logback.version}</version>
</dependency>
```

ZK uses the standard logger [4] to log messages. You could control what to log by configuring the logging of the Web server you are using. The configuration usually varies from one server to another. However, you could use the configuration mechanism provided by ZK as described in this section. It shall work with most Web servers.

There are basically two steps to configure the standard logger with ZK's configuration mechanism:

1. Prepare a logging configuration file
2. Specify the configuration file in a library property

Prepare a logging configuration file

A logging configuration file is a standard properties file. Each line is a key-value pair in the following format:

```
'a.package.or.a.class' = 'level'
```

Here is an example of a configuration file.

```
org.zkoss.zk.ui.impl.UiEngineImpl=FINER
#Make the log level of the specified class to FINER
org.zkoss.zk.ui.http=DEBUG
#Make the log level of the specified package to DEBUG
org.zkoss.zk.ui=OFF
#Turn off the log for the specified package
org.zkoss=WARNING
#Make all log levels of ZK classes to WARNING except those
specified here
```

Allowed Levels

Level	Description
OFF	Indicates no message at all.
SEVERE	Indicates providing error messages.
WARNING	Indicates providing warning messages. It also implies ERROR.
INFO	Indicates providing informational messages. It also implies ERROR and WARNING.
FINE	Indicates providing tracing information for debugging purpose. It also implies ERROR, WARNING and INFO.
FINER	Indicates providing fairly detailed tracing information for debugging purpose. It also implies ERROR, WARNING, INFO and DEBUG

Specify the handler for Jetty and servers that don't turn on the standard logger

Some Web servers, such as Jetty, don't turn on the standard logger by default. Thus, in the logging configuration file, you have to configure the handler too. For example, you can turn on the `java.util.logging.ConsoleHandler` to write the logs to the console by adding the following lines to the logging configuration file:

```
handlers = java.util.logging.ConsoleHandler

java.util.logging.ConsoleHandler.level = FINER
java.util.logging.ConsoleHandler.formatter =
java.util.logging.SimpleFormatter
```

Here is another example that configures the console and a file to be the target of the logs:

```
handlers = java.util.logging.FileHandler,
java.util.logging.ConsoleHandler

java.util.logging.ConsoleHandler.level = FINER
java.util.logging.ConsoleHandler.formatter =
java.util.logging.SimpleFormatter

java.util.logging.FileHandler.formatter =
java.util.logging.SimpleFormatter
java.util.logging.FileHandler.pattern = /var/log/jetty6/solr-%u.log
java.util.logging.FileHandler.level = FINER

org.zkoss.zk.ui.impl.UiEngineImpl=FINER
org.zkoss.bind=FINE
```

Specify the configuration file in a library property

To let ZK load the logging configuration file, you have to specify in a library property called org.zkoss.util.logging.config.file. For example,

```
<library-property>
    <name>org.zkoss.util.logging.config.file</name>
    <value>conf/zk-log.properties</value>
</library-property>
```

If a relative path is specified, it will look for the classpath first. If not found, it will assume it is related to the current directory, i.e., the directory specified in the system property called user.dir.

You could specify an absolute path, such as /usr/jetty/conf/zk-log.properties, if you are not sure what the current directory is.

- [1] <http://www.slf4j.org/>
- [2] <http://www.slf4j.org/manual.html>
- [3] <https://docs.oracle.com/cd/E19717-01/819-7753/gcbkm/index.html>
- [4] <http://docs.oracle.com/javase/1.4.2/docs/guide/util/logging/overview.html>

Disable All Logs

If you want to disable all loggers completely or change the level for all loggers, you don't need to prepare a logging configuration file. Rather, you can configure DHtmlLayoutServlet (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/http/DHtmlLayoutServlet.html#>) in WEB-INF/web.xml as follows.

```
<servlet>
    <servlet-name>zkLoader</servlet-name>
    <servlet-class>org.zkoss.zk.ui.http.DHtmlLayoutServlet</servlet-class>
    <init-param>
        <param-name>log-level</param-name>
        <param-value>OFF</param-value>
    </init-param>
</servlet>
```

For more information, please refer to ZK Configuration Reference.

How to Log

Class: Log (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/logging/Log.html#>)

The logger used by ZK is based on the standard logger, java.util.logging.Logger. However, we wrap it as Log (<http://www.zkoss.org/javadoc/latest/zk/org/zkoss/util/logging/Log.html#>) to make it more efficient.

To log the message to the client rather than the console at the server, you could use Clients.log(java.lang.String) ([http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#log\(java.lang.String\)](http://www.zkoss.org/javadoc/latest/zk/org/zkoss/zk/ui/util/Clients.html#log(java.lang.String)))

The typical use is as follows.

```
import org.zkoss.util.logging.Log;
class MyClass {
    private static final Log log = Log.lookup(MyClass.class);
    public void f(Object v) {
```

```

    if (log.debugable()) log.debug("Value is "+v);
}
}

```

Reference

- Apache Log4j Security Vulnerabilities (<https://logging.apache.org/log4j/2.x/security.html>)

Version History

Version	Date	Content
6.0.0	February 2012	LogService was deprecated.

DSP

Package: org.zkoss.web.servlet.dsp^[1]

A JSP-like template technology. It takes the same syntax as that of JSP. Unlike JSP, DSP is interpreted at the run time, so it is easy to deploy DSP pages. No Java compiler is required in your run-time environment. In addition, you could distribute DSP pages in jar files.

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

If you want to use DSP in your Web applications, you have to set up WEB-INF/web.xml to add the following lines.

```

<servlet>
    <description><! [CDATA[
The servlet loads the DSP pages.
]]></description>
    <servlet-name>dspLoader</servlet-name>
    <servlet-class>org.zkoss.web.servlet.dsp.InterpreterServlet</servlet-class>

    <!-- Specify class-resource, if you want to access TLD defined in jar files -->
    <init-param>
        <param-name>class-resource</param-name>
        <param-value>true</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>dspLoader</servlet-name>
    <url-pattern>*.dsp</url-pattern>
</servlet-mapping>

```

The mapping of the DSP loader is optional. Specify it only if you want to write Web pages in DSP syntax.

Though standard components of ZK use DSP as a template technology, they are handled directly by ZK loader.

Using DSP requires including a jar explicitly, please read ZK Configuration Reference...DSP Loader#Optional_Jar

A Sample of DSP

```
<%@ page contentType="text/css; charset=UTF-8" %>
<%@ taglib uri="http://www.zkoss.org/dsp/web/core" prefix="c" %>

<%-- header.jsp --%>
<style>
<!--Include-->
<c:include page="/css/header.css.dsp" />

<!--Test-->
<c:if test="${c:isSafari() || c:browser('chrome')}">
.search-input-outer input {
    padding: 0 2px;
}
</c:if>

</style>
```

For more details, please check the javadoc of org.zkoss.web.servlet.dsp.action^[2] package.

References

- [1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/web/servlet/dsp/package-summary.html>
- [2] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/web/servlet/dsp/action/package-summary.html>

iDOM

Package: org.zkoss.idom^[1]

An implementation of W3C DOM. It is inspired by JDOM^[2] to have concrete classes for all XML objects, such as Element and Attribute. However, iDOM implements the W3C API, such as org.w3c.dom.Element. Thus, you could use iDOM seamlessly with XML utilities that only accept the W3C DOM.

A typical example is XSLT and XPath. You could use any of favorite XSL processor and XPath utilities with iDOM.

[1] <http://www.zkoss.org/javadoc/latest/zk/org/zkoss/idom/package-summary.html>

[2] <http://www.jdom.org>

Common Error Messages

This section explains those common error messages you might encounter while developing with ZK, so you can figure out what's going wrong and eliminate them.

Your ZK binary is being altered and may not work as expected

It is caused by mixing unexpected jars in your classpath including :

- Mixing different versions of ZK jar (e.g. 8.6.0.1 with 9.6.0.1)
- Mixing enterprise evaluation jar with the official jar (e.g. 9.6.3-Eval with 9.6.3)
- Mixing CE jar with the enterprise evaluation jar

Suggestions

Make sure you use the correct, single source of ZK jar.

- download jar from ZK Maven premium repository
- If you use ZK by copying jar manually, remove all the existing zk jar and copy from a single source again.

If you use Maven, list the final resolved dependencies:

```
mvn dependency:list
```

Then check if there is any mixed-version jar.

The resource you request is no longer available

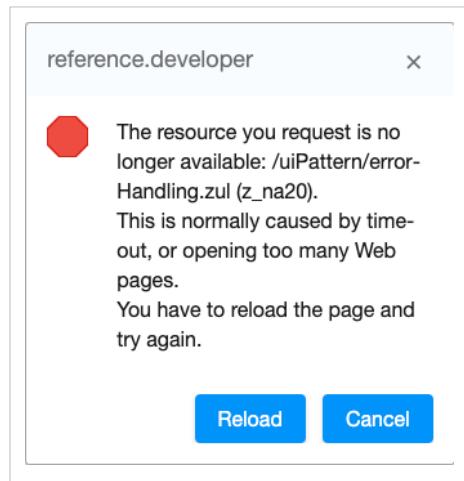
The resource you request is no longer available: /ui/handling.zul (z_na20). This is normally caused by timeout, or opening too many Web pages. You have to reload the page and try again.

Suggestion

It could be caused by

- the corresponding Desktop at the sever is destroyed
- a session is time out
- your server or application is down

Please check them.



Client Error

Severe: [Desktop z_wew:/mypage.zul] client error: Failed to mount: Cannot read property 'colSpan' of undefined

Suggestion

Usually, a failed to mount error is caused by a bug in the specific widget code, invalid custom javascript, missing javascript packages or files, or other causes affecting the client.

When encountering a failed to mount error box, we suggest you to check:

- **Javascript console on a browser**

Open a browser developer tools > console tab. You should find one or more javascript errors in the console. Please expand the error stack trace and provide these full error stacks in the support ticket. This can help us pinpoint the exact cause of the issue.

To make the stack trace readable, you have to enable javascript debug mode in zk.xml

```
<client-config>
<debug-js>true</debug-js>
</client-config>
```

Restart your zk application.

You can enable the send-client-errors element in zk.xml to send client errors to the server for logging the page url where the error occurred and its stack trace.

```
<client-config>
<send-client-errors>true</send-client-errors>
</client-config>
```

- **Network tab on a browser developer tool**

Open a browser developer tools > network tab. ZK does not load every package from page initialization. Instead, packages will be loaded on demand. For example, the zul.inp.wpd (input package) will be loaded if you add an

input component to your page. If a browser was unable to locate or load the relevant package for a newly added widget class, you should see a failed request for this package. This request should also display a return code. Anything other than "200 - success" should be documented and added to the support ticket.

java.lang.IllegalStateException: Access denied: component, <Listcell z_27_b53>, belongs to another desktop: [Desktop g272]

The error message resembles the one mentioned above but with different component names and desktop IDs. This error message is typically caused by an incorrect usage scenario where a setter method of one component is invoked within the event listener of a different desktop (or browser tab).

You can think of one desktop as corresponding to a browser tab, so each component actually belongs to a specific desktop (tab). When you call a setter method like `Window.setTitle()`, it will generate responses to its desktop, but if you call it in another desktop's listener zk will throw this exception.

Please review your code for the following potential issues:

- A variable that references a component in a composer is declared as static
- A component is passed by a session-scope (or application-scope) event queue to another desktop to access
- Retrieving a component from a session (or application) attribute and calling its setter

Suggestions

- You should not declare any variable that references a component as static.
- The purpose of storing (or passing) a component is to communicate with another component.

pass data, not the component

The first principle is: **pass data, not the component**

For example, if you want to get user input from a textbox, don't pass the textbox, pass `textbox.getValue()` instead.

Use Desktop Scope

If you still need to pass a component for later use:

- set an attribute in a Desktop
- pass a component via a Desktop scope event queue

Development-time Best Practices

There are several settings below we recommend you to specify during development time that can improve your developer experience. They can show you more debugging information and disable various caches to get the latest change without restarting a server.

Disable Browser Cache

Since you change pages and codes frequently during development, in order to see the latest change, it's better to disable the browser cache.

Separate zk.xml for Development

Put these development-time settings in a separate file, so that you can easily remove it for production in the future.

zk.xml

```
<library-property>
    <name>org.zkoss.zk.config.path</name>
    <value>/WEB-INF/zk-dev.xml</value>
</library-property>
```

zk-dev.xml

```
<system-config>
    <disable-zscript>false</disable-zscript>
</system-config>

<client-config>
    <debug-js>true</debug-js>
</client-config>
<library-property>
    <name>org.zkoss.zk.ui.versionInfo.enabled</name>
    <value>true</value>
</library-property>

<library-property>
    <name>org.zkoss.zk.ZUML.cache </name>
    <value>false</value>
</library-property>
<library-property>
    <name>org.zkoss.zk.WPD.cache</name>
    <value>false</value>
</library-property>
<library-property>
    <name>org.zkoss.zk.WCS.cache</name>
    <value>false</value>
</library-property>
<library-property>
```

```
<name>org.zkoss.web.classWebResource.cache</name>
<value>false</value>
</library-property>
<library-property>
    <name>org.zkoss.util.label.cache</name>
    <value>false</value>
</library-property>
```

Reloading Modified Java Class without Restarting a Server

Running your application with TravaOpenJDK11 [1] with hot-swap supported JDK can achieve it. Run it with JVM option `-XX:HotswapAgent`. When you modify your java code, re-compile it, the JDK can automatically reload(hot-swap) the new class file that saves the time of restarting an application server.

Use a Server without Packaging and Deploying WAR

Configure Jetty maven plugin [2] in a `pom.xml` can start your project in a few seconds which speeds up the iteration of writing code and testing.

```
<build>
    <plugins>
        <!-- Run with Jetty -->
        <plugin>
            <groupId>org.eclipse.jetty</groupId>
            <artifactId>jetty-maven-plugin</artifactId>
            <version>9.4.27.v20200227</version>
        </plugin>
    </plugins>
</build>
```

Spring Boot

With Spring Boot, please reference this development config [3].

References

- [1] <https://github.com/TravaOpenJDK/trava-jdk-11-dcevm>
- [2] https://wiki.eclipse.org/Jetty/Feature/Jetty_Maven_Plugin
- [3] <https://github.com/zkoss/zkspringboot/blob/master/zkspringboot-demos/zkspringboot-demo-jar/src/main/java/org/zkoss/zkspringboot/demo/DevelopmentConfig.java>

Packaging Applications

Package Single Version in WAR

ZK doesn't support running 2 different versions in a single web application (WAR). Mixing Compact Edition/official version with evaluation version is not allowed. Please be sure you package only one single version of ZK JAR into a WAR.

For example, if you include **zul:9.6.0** and **zkex:9.6.3**, then you will see the following messages when starting a server:

```
2023-03-02 09:21:44 [INFO ] ConfigParser:116 - Ignore
jar:file:/Users/yourName/.m2/repository/org/zkoss/zk/zkex/9.6.3/zkex-9.6.3.jar!/metainfo/
Cause: ZK version must be 9.6.3 or later, not 9.6.0
...
Your ZK binary is being altered and may not work as expected. Please
contact us at info@zkoss.org for assistance.
```

Package ZK As a Shared Library

This is a not-recommended practice. Since ZK stores its configurations (parsed from `zk.xml`) as a static object, all your zk applications in the same application server will have the same configurations. If you change a configuration (library property), it will affect all zk applications.

Package as EAR

An EAR file contains one or more JAR and WAR.

Notice for Form Binding

If you package multiple WAR files and EJB as an EAR file and use form binding on an entity bean packaged in EJB jar, you need to arrange zkbind jar in the following way, or you might encounter the error:

```
java.lang.NoClassDefFoundError: org/zkoss/bind/proxy/FormProxyObject
```

1. put `zkbind-api.jar` and `javassist.jar` in a shared library
2. package `zkbind-impl.jar` in a WAR.
3. make sure a WAR doesn't contain `zkbind-api.jar`

Check this EAR demo project ^[1] for details.

myapp.ear

```
lib/javassist.jar
lib/zkbind-api.jar
lib/zk.jar (and its dependent jar)
```

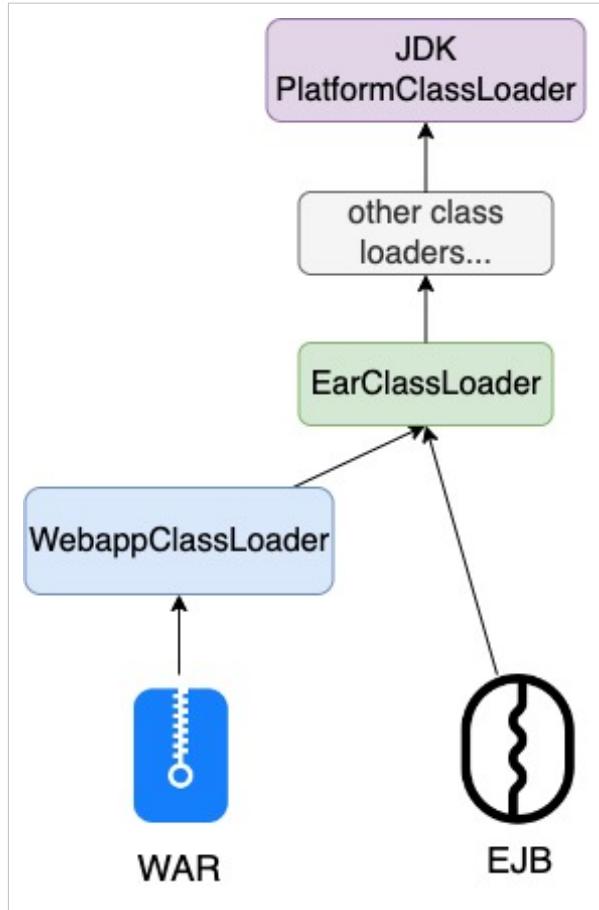
```
myapp-ejb.jar (MyEntity.java)
```

```
myapp.war/WEB-INF/lib/zkbind-impl.jar
myapp.war/WEB-INF/lib/other ZK jars (but NO zkbind-api.jar here)
```

If you don't use form binding or don't use form binding on a bean in an EJB jar, then you don't need above arrangement.

Underlying Details

In enterprise application server, each module has its own class loader like:



When ZK creates a proxy object based on EJB class with javassist, it uses the target object's class loader. In this case, it's EJB's class loader.

The related code in javassist:3.28.0-GA

```

public class ProxyFactory {
    ...
    protected ClassLoader getClassLoader0() {
        ClassLoader loader = null;
        if (superClass != null &&
!superClass.getName().equals("java.lang.Object"))
            loader = superClass.getClassLoader();
    }
}
  
```

- Line 6: superClass is the EJB bean to be proxied.

Since parent class loader cannot know the classes loaded by its child class loader, EarClassLoader doesn't know the classes loaded by WebappClassLoader. So it will throw `java.lang.NoClassDefFoundError: org/zkoss/bind/proxy/FormProxyObject` because zkbind.jar is in a WAR.

Hence, that's the reason you need to put zkbind-api.jar in /lib (out of WAR) in order to be loaded by EarLibClassLoader.

(Related issues: ZK-2932^[2], ZK-3135^[3], ZK-4554^[4], ZK-5256^[5])

Class Loader Hierarchy in Glassfish 6

WAR

WebappClassLoader (load classes under WEB-INF) |||
EarClassLoader |||
org.glassfish.internal.api.DelegatingClassLoader@6144db94 |||
EarLibClassLoader (load jar in lib) |||
org.glassfish.internal.api.DelegatingClassLoader@76bcc8a2 |||
java.net.URLClassLoader@42561fba |||
APIClassLoader |||
jdk.internal.loader.ClassLoaders\$PlatformClassLoader@80937

EJB

EarClassLoader |||
org.glassfish.internal.api.DelegatingClassLoader@6144db94 |||
EarLibClassLoader |||
org.glassfish.internal.api.DelegatingClassLoader@76bcc8a2 |||
java.net.URLClassLoader@42561fba |||
APIClassLoader |||
jdk.internal.loader.ClassLoaders\$PlatformClassLoader@80937

References

- [1] <https://github.com/hawkchen/ear-demo>
 - [2] <https://tracker.zkoss.org/browse/ZK-2932>
 - [3] <https://tracker.zkoss.org/browse/ZK-3135>
 - [4] <https://tracker.zkoss.org/browse/ZK-4554>
 - [5] <https://tracker.zkoss.org/browse/ZK-5256>
-

How to get Efficient Support

Support Channel

If you are a ZK customer and your package comes with remote consulting hours, you can contact mailto:support@potix.com to request technical support. Please include your Invoice Number or License Certificate number when you submit your question so that we can associate you with the right company.

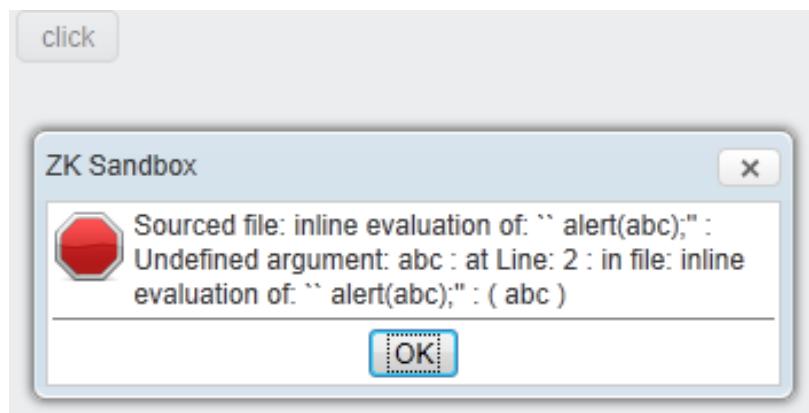
If you do not have remote consulting hours or if you are not sure, please read Customer FAQ ^[1] or contact us at info@zkoss.org.

Find Known Issues ^[2]

If you think you encounter a bug, you can search for the bug in our tracker ^[3]. If it's a known bug, there is sometimes a workaround mentioned in the tracker. You can apply the workaround before you get the official release.

Getting prompt and efficient support from ZK

When you run into an issue in ZK, following this document can help you to provide the support team with sufficient data, thus speeding up the investigation and get prompt and efficient feedback. Here is a sample screenshot of an error. However, with only a screenshot, it is difficult for us to find enough clues and provide timely support. Instead of providing a screenshot, please follow the tips and instructions below when submitting your issue.



Provide Sufficient information

Providing sufficient information helps us support you more efficiently.

The Sufficient Information Includes:

1. the complete server-side error or the client-side error if any (read #Check JavaScript Error)
2. complete exception stack trace (if any)
3. A reproducible sample (please read #Tips for creating a reproducible sample)
4. Precise and clear steps to reproduce the issue
5. ZK version and edition
e.g. ZK 9.0.1 PE
6. Browser and its version
e.g. Firefox 60

7. If it is hard to create a sample, you can provide an URL so that the support team can access it. When providing this URL, remember to turn on the js debug ^[4] in your zk.xml.
8. Application servers (e.g. JBoss 6, if it's necessary).
9. If your case happens only in your specific environment, it is recommended to request an online meeting or provide a virtual machine as a copy of your actual environment
10. anything else that can help us understand your issue better

Tips for creating a reproducible sample

You can provide:

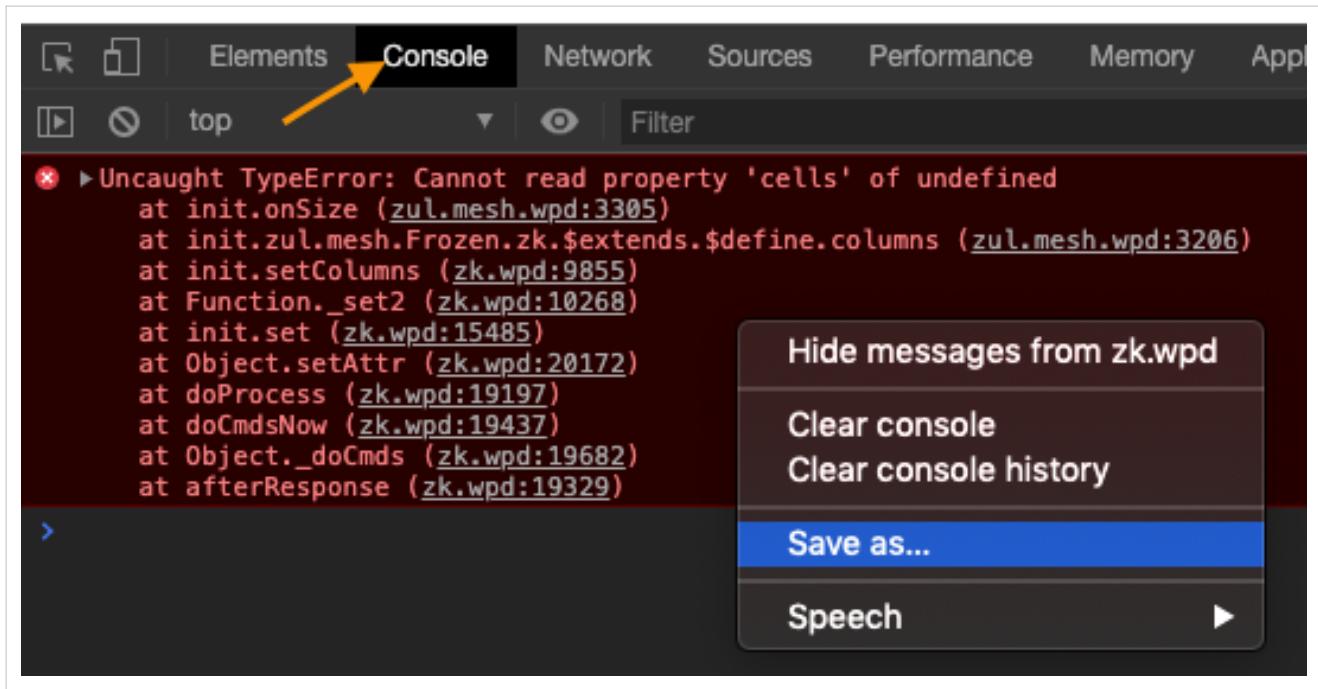
1. Use zkfiddle ^[5]
2. Zip your maven project with pom.xml and include necessary source code
3. Provide zul and its related Java source
4. Provide a public URL that we can visit

Steps:

1. Clone the original zul with the issue and remove irrelevant components.
2. Replace your database data with static sample data in order to run without a database.

Check JavaScript Error

You can check if there is any JavaScript error by opening the developer tool. For example, press F12 to open the chrome developer tool/ Console tab:



You can extract the error message by:

- select and copy the whole text or
- right-click, select "Save as..."

If you cannot provide a reproducible sample

Then you can choose the following alternatives:

Provide a Public URL of Your Web Application

Provide an URL so that the support team can access it. When providing this URL, remember to turn on the js debug in your zk.xml.

Provide ZK AU Request/Response Details

1. Open the browser developer tool before starting to reproduce your issue
2. Do the steps to reproduce your problem
3. Send request details as HAR to us

* Save all network requests to a HAR file in Chrome [6]

* Save request details as HAR in Firefox

The screenshot shows the Firefox developer tools Network tab. A context menu is open over a list of network requests. The menu options are: Copy, Save All As HAR (which is highlighted in blue), Save Image As, Resend, and Edit and Resend. The requests listed are: 200 GET localhost:8080, 200 GET localhost:8080, 200 GET www.zkoss.org, and 200 POST localhost:8080.

* Save request details as HAR in Edge [7]

If your browser doesn't support saving requests as HAR, or just 1 request is related, you can also send us the screenshot of AU request like:

The screenshot shows the Chrome developer tools Network tab. A specific request named 'zkau' is selected. The payload tab is active, showing the following data:
 dtid: z_lMQ4_WuNPF2n4bJ9X6gxDw
 cmd_0: onClick
 uuid_0: vV9P2
 data_0: {"pageX":96,"pageY":54,"which":1,"x":24,"y":21}

Provide Related Source

By checking your zul and Java source, we can try to simulate your case. This is better than inferring what you describe in natural language.

How to Include a JavaScript file

When you need to apply a js patch or include a 3rd-party js library, you have to include the js file in your application. This section describes how you can include the js file for different scopes.

Page-Scope

- Use the script directive <?script?>
- Use the script component <script>

Application-Scope

If you need to include a javascript file on every zul, there are 2 ways:

By language addon

Create a `lang-addon.xml` according to ZK Client-side Reference/Language Definition#Language_Addon and include the javascript file with <javascript>. For example:

```
<language-addon>
    <addon-name>patch-addon</addon-name><!-- give a meaningful name -->
    <language-name>xul/html</language-name>
    <depends>zul</depends>
    <javascript src="~./mypatch.js" />
    <javascript src="/zkpatch/mypatch2.js"/>
</language-addon>
```

- Line 4: see the next section
- Line 5: a path starting with `~./` is a classpath web resource path which is a special path supported by ZK
- Line 6: You can also link a file under your web application context root

Dependent Addon

If you override a component's widget or extend an existing component, it's crucial to specify <depends> correctly, so that your lang addon will take effect. According to which component you override, you need to specify the corresponding addon name. For example:

- If you override a component in zul language e.g. <button>, you specify
`<depends>zul</depends>`
- If you override a component provided by zkmax e.g. <nav>, you specify
`<depends>zkmax</depends>`
- If you override something about accessibility, e.g. override aria attribute, you specify
`<depends>zally</depends>`

How to Run a Sample Maven Project

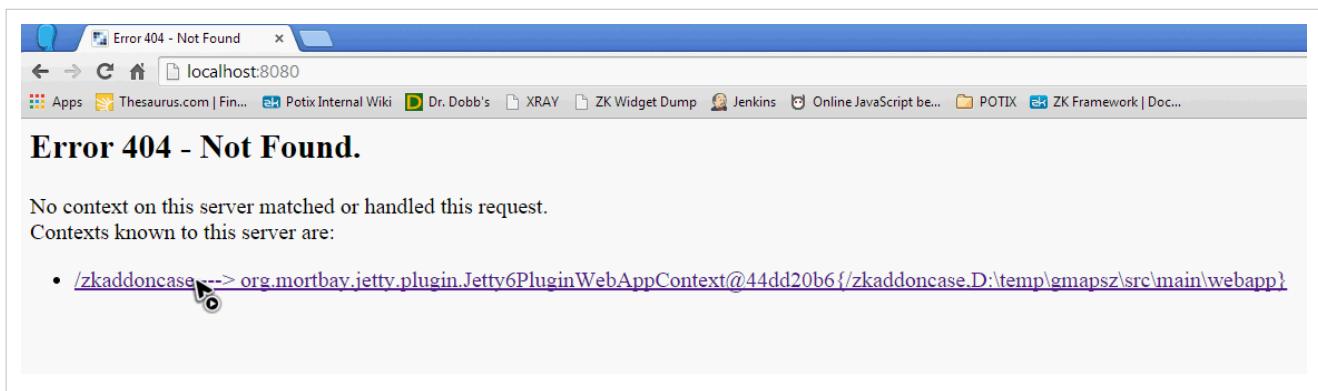
In order to narrow down the problem and eliminate differences caused by different environments, we often send you a sample maven project that tries to simulate and reproduce your issue. To run the project, you need to download maven [8] and setup [9] first.

1. Run your command line interface
2. switch to the sample project's root folder that has a `pom.xml`
3. Start jetty server with the command below

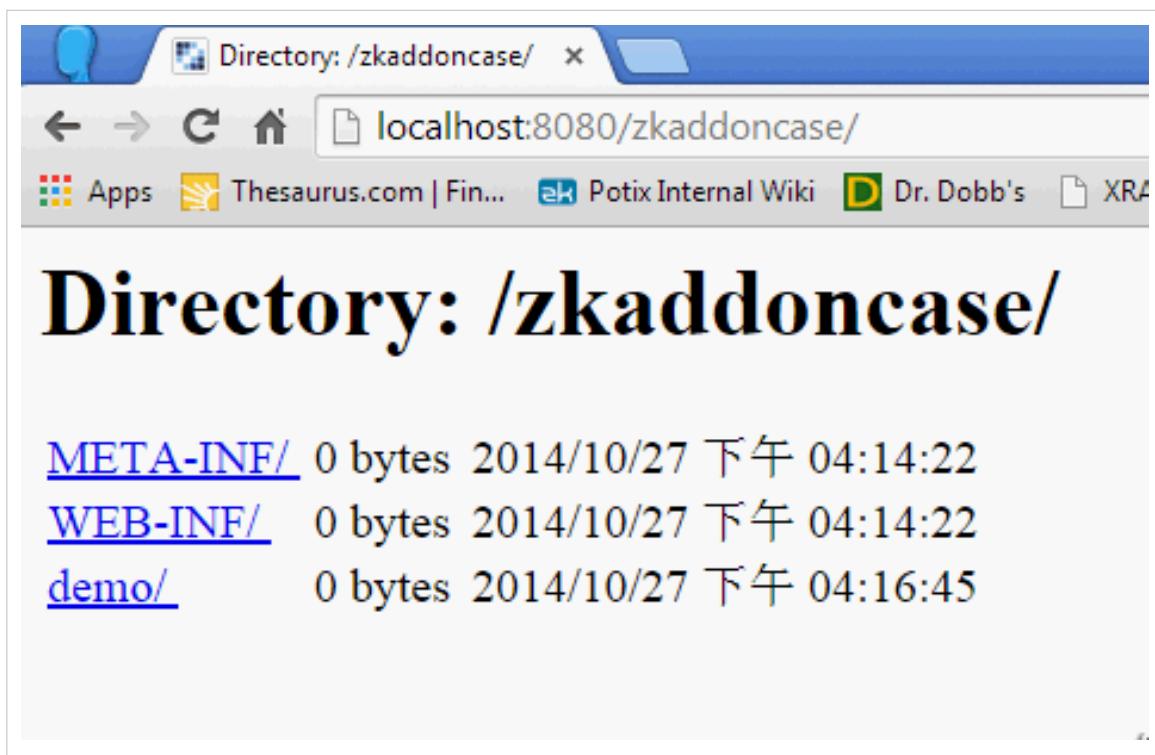
```
mvn jetty:run
```

4. Visit the project's pages with your browser

If you don't know the zul's path or if the path is incorrect, just visit `http://localhost:8080`. A browser will show a link with the correct context path:



Click the link and it will list folders then you can navigate to find the zul.



How to get Browser Performance Profile

If you are running into a performance issue, you can first check out Step by Step Trouble Shooting ^[10] and do a first step analysis using browser's developer tools.

References

- [1] https://www.zkoss.org/wiki/Customers/FAQ#What_is_my_current_ZK_Package.3F
- [2] <https://tracker.zkoss.org/issues/>
- [3] <https://tracker.zkoss.org/issues>
- [4] <https://www.zkoss.org/wiki/ZK%20Configuration%20Reference/zk.xml/The%20client-config%20Element/The%20debug-js%20Element>
- [5] <http://zkfiddle.org>
- [6] <https://developers.google.com/web/tools/chrome-devtools/network/reference#save-as-har>
- [7] <https://docs.microsoft.com/en-us/microsoft-edge/devtools-guide/network>
- [8] <http://maven.apache.org/download.cgi>
- [9] <http://maven.apache.org/guides/getting-started/maven-in-five-minutes.html>
- [10] https://www.zkoss.org/wiki/ZK_Developer's_Reference/Performance_Monitoring/Step_by_Step_Trouble_Shooting

Article Sources and Contributors

ZK Developer's Reference *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference *Contributors:* Alicelin, Hawk, Jeanher, Southerncrossie, Sphota, Tmillsclare, Tomyeh, Wingor

Overture *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Overture *Contributors:* Alicelin, Hawk, Samchuang, Tomyeh

Architecture Overview *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Overture/Architecture_Overview *Contributors:* Alicelin, Char, Flyworld, Hawk, Neillee, RebeccaLai, Tomyeh

Technology Guidelines *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Overture/Technology_Guidelines *Contributors:* Alicelin, Hawk, Iantsai, Raymondchao, RebeccaLai, SimonPai, Tmillsclare, Tomyeh

Extensions *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Overture/Extensions *Contributors:* Alicelin, Chanwit, Hawk, Jumperchen, RebeccaLai, SimonPai, Tomyeh

Example Project *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Overture/Example_Project *Contributors:* Hawk, RebeccaLai

UI Composing *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Composing *Contributors:* Alicelin, Tomyeh

Component-based UI *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Composing/Component-based_UI *Contributors:* Alicelin, Gap77, Hawk, Henrichen, Neillee2, RebeccaLai, Tomyeh

ID Space *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Composing/ID_Space *Contributors:* Alicelin, Hawk, Henrichen, M17, RebeccaLai, Tomyeh, Vincent

ZUML *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Composing/ZUML *Contributors:* Alicelin, Char, Hawk, RebeccaLai, SimonPai, Tomyeh

XML Background *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Composing/ZUML/XML_Background *Contributors:* Alicelin, Hawk, RebeccaLai, Tomyeh

Basic Rules *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Composing/ZUML/Basic_Rules *Contributors:* Alicelin, Hawk, RebeccaLai, Tomyeh

EL Expressions *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Composing/ZUML/EL_Expressions *Contributors:* Alicelin, Dark1sun, Hawk, Matthewcheng, RebeccaLai, Robertwenzel, Tomyeh

Scripts in ZUML *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Composing/ZUML/Scripts_in_ZUML *Contributors:* Alicelin, Hawk, RebeccaLai, Tomyeh

Conditional Evaluation *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Composing/ZUML/Conditional_Evaluation *Contributors:* Alicelin, Hawk, RebeccaLai, Tomyeh

Iterative Evaluation *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Composing/ZUML/Iterative_Evaluation *Contributors:* Alicelin, Ashishd, Hawk, RebeccaLai, Tomyeh, Vincent

On-demand Evaluation *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Composing/ZUML/On-demand_Evaluation *Contributors:* Alicelin, Hawk, Jumperchen, RebeccaLai, Tomyeh

Include a Page *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Composing/ZUML/Include_a_Page *Contributors:* Hawk, Jameschu, RebeccaLai

Load ZUML in Java *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Composing/ZUML/Load_ZUML_in_Java *Contributors:* Alicelin, Ashishd, Char, Hawk, Henrichen, RebeccaLai, Robertwenzel, Tomyeh

XML Namespaces *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Composing/ZUML/XML_Namespaces *Contributors:* Alicelin, Hawk, RebeccaLai, Tomyeh

Richlet *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Composing/Richlet *Contributors:* Alicelin, Char, Hawk, Jumperchen, M17, RebeccaLai, Robertwenzel, Sphota, Tomyeh

Macro Component *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Composing/Macro_Component *Contributors:* Alicelin, Char, Hawk, RebeccaLai, Tomyeh

Inline Macros *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Composing/Macro_Component/Inline_Macros *Contributors:* Alicelin, Hawk, RebeccaLai, Tomyeh

Implement Custom Java Class *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Composing/Macro_Component/Implement_Custom_Java_Class *Contributors:* Alicelin, Char, Hawk, RebeccaLai, Robertwenzel, Southerncrossie, Tomyeh, Vincent

Composite Component *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Composing/Composite_Component *Contributors:* Alicelin, Char, Hawk, Peterkuo, RebeccaLai, Tomyeh

Client-side UI Composing *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Composing/Client-side_UI_Composing *Contributors:* Alicelin, Hawk, Tomyeh

Shadow for MVC *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Composing/Shadow_for_MVC *Contributors:* Chunfuchang, Hawk, RebeccaLai, Tendysu

Event Handling *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Event_Handling *Contributors:* Phoebelin, RebeccaLai, Tomyeh

Event Listening *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Event_Handling/Event_Listening *Contributors:* Alicelin, Ashishd, Hawk, Henrichen, Kbsimm, RebeccaLai, Robertwenzel, Tomyeh

Event Firing *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Event_Handling/Event_Firing *Contributors:* Alicelin, Hawk, Peterkuo, RebeccaLai, Tomyeh

Event Forwarding *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Event_Handling/Event_Forwarding *Contributors:* Alicelin, Ashishd, Flyworld, Hawk, Henrichen, RebeccaLai, Tomyeh, Wenninghsu

Event Queues *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Event_Handling/Event_Queue *Contributors:* Alicelin, Ashishd, Hawk, Henrichen, Jeanher, Jumperchen, RebeccaLai, Tmillsclare, Tomyeh, Vincent

Client-side Event Listening *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Event_Handling/Client-side_Event_Listening *Contributors:* Alicelin, Hawk, RebeccaLai, Tomyeh, Wenninghsu

MVC *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC *Contributors:* Hawk, Henrichen, Raymondchao, RebeccaLai, Tomyeh

Controller *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/Controller *Contributors:* Alicelin, Char, Henrichen, RebeccaLai, Tomyeh

Composer *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/Controller/Composer *Contributors:* Alicelin, Dennischen, Hawk, Henrichen, Iantsai, M17, Raymondchao, RebeccaLai, SimonPai, Tomyeh, Wenninghsu

Wire Components *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/Controller/Wire_Components *Contributors:* Christopherszu, Hawk, Raymondchao, RebeccaLai, Tomyeh, Wenninghsu

Wire Variables *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/Controller/Wire_Variables *Contributors:* Alicelin, Char, Hawk, Jimmyshiau, Matthieu, RebeccaLai, Tomyeh

Wire Event Listeners *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/Controller/Wire_Event_Listeners *Contributors:* Alicelin, Char, Hawk, RebeccaLai, SimonPai, Tomyeh, Wenninghsu

Subscribe to EventQueues *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/Controller/Subscribe_to_EventQueues *Contributors:* Hawk, Jeanher, Jerrychen, RebeccaLai, Samchuang, SimonPai, Southerncrossie

Model *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/Model *Contributors:* Alicelin, Hawk, MontyPan, RebeccaLai, Tomyeh

List Model *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/Model/List_Model *Contributors:* Alicelin, Hawk, Jumperchen, RebeccaLai, SimonPai, Tendysu, Tomyeh

Groups Model *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/Model/Groups_Model *Contributors:* Alicelin, Francishtiao, Hawk, Jimmyshiau, Jumperchen, Matthieu, RebeccaLai, Tomyeh

Tree Model *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/Model/Tree_Model *Contributors:* Alicelin, Hawk, Jameschu, Jimmyshiau, Jumperchen, MontyPan, RebeccaLai, Robertwenzel, Simbo, SimonPai, Southerncrossie, Tomyeh, Vincent

Chart Model *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/Model/Chart_Model *Contributors:* Alicelin, Hawk, Raymondchao, RebeccaLai, Tomyeh

Matrix Model *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/Model/Matrix_Model *Contributors:* Hawk, Jumperchen, RebeccaLai

View *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/View *Contributors:* Alicelin, Hawk, RebeccaLai, Tomyeh

Template *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/View/Template *Contributors:* Hawk, Raymondchao, Tomyeh

Listbox Template *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/View/Template/Listbox_Template *Contributors:* Hawk, Henrichen, Jumperchen, Raymondchao, RebeccaLai, Tmillsclare, Tomyeh

Grid Template *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/View/Template/Grid_Template *Contributors:* Hawk, Jumperchen, RebeccaLai

Tree Template *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/View/Template/Tree_Template *Contributors:* Hawk, Tomyeh

Combobox Template *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/View/Template/Combobox_Template *Contributors:* Hawk, Tomyeh

Selectbox Template *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/View/Template/Selectbox_Template *Contributors:* Hawk, Tomyeh

Biglistbox Template *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/View/Template/Biglistbox_Template *Contributors:* Hawk, Jumperchen, Wenninghsu

Chosenbox Template *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/View/Template/Chosenbox_Template *Contributors:* Benbai, Hawk

Tabbox Template *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/View/Template/Tabbox_Template *Contributors:* Hawk, Jumperchen, Robertwenzel

Organigram Template *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/View/Template/Organigram_Template *Contributors:* Charlesqiu, Hawk

Searchbox Template *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/View/Template/Searchbox_Template *Contributors:* Hawk, Rudyhuang

Renderer *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/View/Renderer *Contributors:* Hawk, RebeccaLai, Tomyeh

Listbox Renderer *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/View/Renderer/Listbox_Renderer *Contributors:* Hawk, Jumperchen, Tomyeh

Grid Renderer *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/View/Renderer/Grid_Renderer *Contributors:* Hawk, Tomyeh

Tree Renderer *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/View/Renderer/Tree_Renderer *Contributors:* Hawk, Tomyeh

Combobox Renderer *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/View/Renderer/Combobox_Renderer *Contributors:* Hawk, Tomyeh

Tabbox Renderer *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/View/Renderer/Tabbox_Renderer *Contributors:* Hawk, Jumperchen

Organigram Renderer *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/View/Renderer/Organigram_Renderer *Contributors:* Charlesqiu, Hawk

Biglistbox Renderer *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/View/Renderer/Biglistbox_Renderer *Contributors:* Hawk, Jumperchen

Item Renderer *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/MVC/View/Renderer/Item_Renderer *Contributors:* Hawk

Stateless Components *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Stateless_Components *Contributors:* Hawk

Building Stateless UI *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Stateless_Components/Building_Stateless_UI *Contributors:* Hawk

Annotations *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Annotations *Contributors:* Alicelin, Tomyeh

Annotate in ZUML *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Annotations/Annotate_in_ZUML *Contributors:* Alicelin, Hawk, Henrichen, RebeccaLai, Tomyeh

Annotate in Java *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Annotations/Annotate_in_Java *Contributors:* Alicelin, Hawk, Tomyeh

Retrieve Annotations *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Annotations/Retrieve_Annotations *Contributors:* Hawk, Tomyeh

Annotate Component Definitions *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Annotations/Annotate_Component_Definitions *Contributors:* Alicelin, Hawk, Tomyeh

UI Patterns *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns *Contributors:* Alicelin, Tomyeh

Responsive Design *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Responsive_Design *Contributors:* Hawk, Jumperchen, RebeccaLai

Message Box *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Message_Box *Contributors:* Hawk, RebeccaLai, Tomyeh

Layouts and Containers *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Layouts_and_Containers *Contributors:* Alicelin, Hawk, Jimmyshiau, RebeccaLai, Southerncrossie, Tomyeh

Hflex and Vflex *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Hflex_and_Vflex *Contributors:* Alicelin, Hawk, Jimmyshiau, Peterkuo, RebeccaLai, SimonPai, Southerncrossie, Tomyeh, Vincent

Grid's Columns and Hflex *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Grid%27s_Columns_and_Hflex *Contributors:* Alicelin, Flyworld, Hawk, Henrichen, Jimmyshiau, RebeccaLai, SimonPai, Tmillsclare, Tomyeh, Vincent

Tooltips, Context Menus and Popups *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Toolips%2C_Context_Menus_and_Popups *Contributors:* Alicelin, Hawk, Jimmyshiau, MontyPan, Raymondchao, RebeccaLai, Rudyhuang, Southerncrossie, Tomyeh, Vincent

Keystroke Handling *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Keystroke_Handling *Contributors:* Alicelin, Hawk, Jameschu, Matthieu, Peterkuo, RebeccaLai, Robertwenzel, Tmillsclare, Tomyeh

Drag and Drop *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Drag_and_Drop *Contributors:* Alicelin, Hawk, RebeccaLai, Tomyeh

Page Initialization *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Page_Initialization *Contributors:* Alicelin, Flyworld, Hawk, Robertwenzel, Sphota, Tomyeh, Vincent

Forward and Redirect *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Forward_and_Redirect *Contributors:* Alicelin, Hawk, Matthieu, RebeccaLai, Tomyeh, V0v87, Vincent

File Upload and Download *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/File_Upload_and_Download *Contributors:* Hawk, RebeccaLai, Tomyeh

Browser Information and Control *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Browser_Information_and_Control *Contributors:* Alicelin, Hawk, RebeccaLai, Robertwenzel, Southerncrossie, Tomyeh, Vincent

Browser History Management *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Browser_History_Management *Contributors:* Alicelin, Flyworld, Hawk, RebeccaLai, Tomyeh

Session Timeout Management *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Session_Timeout_Management *Contributors:* Alicelin, Ashishd, Benbai, Hawk, Matthewcheng, RebeccaLai, Tomyeh

Error Handling *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Error_Handling *Contributors:* Alicelin, Ashishd, Hawk, Iantsai, RebeccaLai, Robertwenzel, Rudyhuang, Tomyeh

Actions and Effects *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Actions_and_Effects *Contributors:* Alicelin, Hawk, Matthewcheng, RebeccaLai, Tomyeh

Useful Java Utilities *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Useful_Java_Utils *Contributors:* Hawk, Jameschu, Jeanher, Jumperchen, Leonlee, MontyPan, RebeccaLai, Rudyhuang, SimonPai, Vincent

HTML Tags *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/HTML_Tags *Contributors:* Alicelin, Hawk, RebeccaLai, Tomyeh

The html Component *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/HTML_Tags/The_html_Component *Contributors:* Char, Hawk, Tomyeh

The native Namespace *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/HTML_Tags/The_native_Namespace *Contributors:* Alicelin, Char, Hawk, RebeccaLai, Tomyeh

The XHTML Component Set *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/HTML_Tags/The_XHTML_Component_Set *Contributors:* Alicelin, Char, Hawk, Jameschu, RebeccaLai, Tomyeh

Long Operations *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Long_Operations *Contributors:* RebeccaLai, Tomyeh

Use Echo Events *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Long_Operations/Use_Echo_Events *Contributors:* Alicelin, Hawk, Jimmyshiau, RebeccaLai, Sphota, Tomyeh

Use Event Queues *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Long_Operations/Use_Event_Queues *Contributors:* Alicelin, Hawk, RebeccaLai, Tomyeh

Use Piggyback *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Long_Operations/Use_Piggyback *Contributors:* Alicelin, Hawk, Raymondchao, RebeccaLai, Tomyeh

Communication *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Communication *Contributors:* RebeccaLai, Tomyeh

Inter-Page Communication *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Communication/Inter-Page_Communication *Contributors:* Alicelin, Hawk, RebeccaLai, Tomyeh

Inter-Desktop Communication *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Communication/Inter-Desktop_Communication *Contributors:* Alicelin, Hawk, RebeccaLai, Tomyeh

Inter-Application Communication *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Communication/Inter-Application_Communication *Contributors:* Alicelin, Hawk, Robertwenzel, Tomyeh

Templating *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Templating *Contributors:* Alicelin, Dennischen, Tomyeh

Composition *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Templating/Composition *Contributors:* Alicelin, Dennischen, Hawk, Jameschu, Jumperchen, RebeccaLai, Tomyeh

Templates *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Templating/Templates *Contributors:* Hawk, RebeccaLai, Tomyeh

XML Output *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/XML_Output *Contributors:* Alicelin, Hawk, RebeccaLai, Tomyeh

Event Threads *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Event_Threads *Contributors:* Alicelin, Hawk, Jumperchen, Matthieu, Maya001122, Tomyeh

Modal Windows *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Event_Threads/Modal_Windows *Contributors:* Alicelin, Char, Hawk, Jimmyshiau, Tomyeh

Message Box *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Event_Threads/Message_Box *Contributors:* Char, Hawk, Tomyeh

File Upload *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/UI_Patterns/Event_Threads/File_Upload *Contributors:* Char, Dennischen, Hawk, Tomyeh, Tonyq

Theming and Styling *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/TheMING_and_Styling *Contributors:* Alicelin, Hawk, Jeanher, Neillie2, Tomyeh

Molds *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/TheMING_and_Styling/Molds *Contributors:* Alicelin, Hawk, RebeccaLai, Tomyeh

CSS Classes and Styles *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/TheMING_and_Styling/CSS_Classes_and_Styles *Contributors:* Alicelin, Hawk, RebeccaLai, Tomyeh

ZK Official Themes *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/TheMING_and_Styling/ZK_Official_Themes *Contributors:* Hawk, Jeanher, RebeccaLai, Robertwenzel, Southerncrossie, Vincent

Switching Themes *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/TheMING_and_Styling/Switching_Themes *Contributors:* Hawk

Customizing Standard Themes *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/TheMING_and_Styling/Customizing_Standard_Themes *Contributors:* Alicelin, Hawk, Jeanher, Neillie2, RebeccaLai, Robertwenzel, SimonPai, Tomyeh, Vincent

Creating Custom Themes *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/TheMING_and_Styling/Creating_Custom_Themes *Contributors:* Hawk, Neillie2

Theme Template *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/TheMING_and_Styling/Creating_Custom_Themes/Theme_Template *Contributors:* Hawk

Archive-based Themes *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/TheMING_and_Styling/Creating_Custom_Themes/Archive-based_Themes *Contributors:* Hawk, Neillee2, Vincent

Folder-based Themes *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/TheMING_and_Styling/Creating_Custom_Themes/Folder-based_Themes *Contributors:* Hawk, Neillee2, Vincent

Understanding the Theming Subsystem *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/TheMING_and_Styling/Understanding_the_Theming_Subsystem *Contributors:* Neillee2, RebeccaLai

Information about a Theme *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/TheMING_and_Styling/Understanding_the_Theming_Subsystem/Information_about_a_Theme *Contributors:* Hawk, Neillee2, RebeccaLai

Registering your Theme *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/TheMING_and_Styling/Understanding_the_Theming_Subsystem/Registering_your_Theme *Contributors:* Hawk, Neillee2, RebeccaLai

Providing Theme Resources *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/TheMING_and_Styling/Understanding_the_Theming_Subsystem/Providing_Theme_Resources *Contributors:* Alicelin, Char, Hawk, Neillee2, Tomyeh, Vincent

Resolving Theme URLs *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/TheMING_and_Styling/Understanding_the_Theming_Subsystem/Resolving_Theme_URLs *Contributors:* Neillee2

Internationalization *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Internationalization *Contributors:* Tmillsclare, Tomyeh

Locale *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Internationalization/Locale *Contributors:* Alicelin, Char, Hawk, Maya001122, RebeccaLai, Samchuang, Tomyeh

Time Zone *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Internationalization/Time_Zone *Contributors:* Alicelin, Hawk, Maya001122, RebeccaLai, Tomyeh

Handling Server and User Time Zones *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Internationalization/Time_Zone/Handling_Server_and_User_Time_Zones *Contributors:* Hawk, Jeanher, RebeccaLai

Labels *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Internationalization/Labels *Contributors:* Alicelin, Char, Hawk, Jimmyshiau, Matthieu, Maya001122, RebeccaLai, Robertwenzel, SimonPai, Tomyeh, Tonyq

The Format of Properties Files *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Internationalization/Labels/The_Format_of_Properties_Files *Contributors:* Alicelin, Ashishd, Hawk, RebeccaLai, Tomyeh

Date and Time Formatting *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Internationalization/Date_and_Time_Formatting *Contributors:* Alicelin, Hawk, Jumperchen, RebeccaLai, Tomyeh

The First Day of the Week *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Internationalization/The_First_Day_of_the_Week *Contributors:* Alicelin, Char, Hawk, Maya001122, RebeccaLai, Tomyeh

Locale-Dependent Resources *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Internationalization/Locale-Dependent_Resources *Contributors:* Alicelin, Char, Hawk, Maya001122, RebeccaLai, Tomyeh

Warning and Error Messages *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Internationalization/Warning_and_Error_Messages *Contributors:* Ashishd, Char, Hawk, Jimmyshiau, Matthieu, Maya001122, RebeccaLai, Robertwenzel, Tomyeh

Server Push *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Server_Push *Contributors:* Alicelin, Hawk, Sphota, Tomyeh

Event Queues *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Server_Push/Event_Queue *Contributors:* Alicelin, Hawk, RebeccaLai, Tomyeh

Synchronous Tasks *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Server_Push/Synchronous_Tasks *Contributors:* Hawk, RebeccaLai, Tomyeh

Asynchronous Tasks *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Server_Push/Asynchronous_Tasks *Contributors:* Alicelin, Hawk, Tomyeh

Configuration *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Server_Push/Configuration *Contributors:* Alicelin, Christopherszu, Hawk, Jeanher, Jimmyshiau, RebeccaLai, Robertwenzel, Sefilin, Tomyeh, Vincent, Wenninghsu

Clustering *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Clustering *Contributors:* Hawk, Tomyeh

ZK Configuration *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Clustering/ZK_Configuration *Contributors:* Alicelin, Hawk, Jeanher, Jimmyshiau, RebeccaLai, Robertwenzel, Tomyeh, Vincent

Server Configuration *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Clustering/Server_Configuration *Contributors:* Hawk, RebeccaLai, Tomyeh, Vincent

Programming Tips *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Clustering/Programming_Tips *Contributors:* Alicelin, Ashishd, Hawk, Peterkuo, RebeccaLai, Tomyeh

Integration *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Integration *Contributors:* Hawk, Tomyeh

Accessing Java EE Scope Objects *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Integration/Accessing_Java_EE_Scope_Objects *Contributors:* Hawk

Presentation Layer *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Integration/Presentation_Layer *Contributors:* Hawk, Southerncrossie

Bootstrap *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Integration/Presentation_Layer/Bootstrap *Contributors:* Hawk

JSP *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Integration/Presentation_Layer/JSP *Contributors:* Alicelin, Dennischen, Hawk, RebeccaLai, Tmillsclare, Tomyeh

Struts *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Integration/Presentation_Layer/Struts *Contributors:* Alicelin, Hawk, Tomyeh

Portal *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Integration/Presentation_Layer/Portal *Contributors:* Alicelin, Hawk, Tomyeh

ZK Filter *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Integration/Presentation_Layer/ZK_Filter *Contributors:* Hawk, RebeccaLai, Tomyeh

Foreign Templating Framework *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Integration/Presentation_Layer/Foreign_Template_Framework *Contributors:* Alicelin, Hawk, Tomyeh

Middleware Layer *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Integration/Middleware_Layer *Contributors:* Hawk, RebeccaLai

Spring *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Integration/Middleware_Layer/Spring *Contributors:* Alicelin, Flyworld, Hawk, Henrichen, Tomyeh

CDI *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Integration/Middleware_Layer/CDI *Contributors:* Alicelin, Hawk, Paowang, Tomyeh

EJB *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Integration/Middleware_Layer/EJB *Contributors:* Alicelin, Hawk, Tomyeh

Persistence Layer *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Integration/Persistence_Layer *Contributors:* Hawk

JDBC *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Integration/Persistence_Layer/JDBC *Contributors:* Alicelin, Hawk, Matthewcheng, Tomyeh

Hibernate *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Integration/Persistence_Layer/Hibernate *Contributors:* Alicelin, Ashishd, Hawk, Henrichen, Matthewcheng, Tomyeh

JPA *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Integration/Persistence_Layer/JPA *Contributors:* Hawk

Security *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Integration/Security *Contributors:* Hawk

Spring Security *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Integration/Security/Spring_Security *Contributors:* Hawk, Matthieu, Robertwenzel

Miscellaneous *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Integration/Miscellaneous *Contributors:* Hawk, Southerncrossie

Google Analytics *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Integration/Miscellaneous/Google_Analytics *Contributors:* Hawk, Tomyeh

Start Execution in Foreign Ajax Channel *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Integration/Miscellaneous/Start_Execution_in_Foreign_Ajax_Channel *Contributors:* Alicelin, Ashishd, Hawk, Henrichen, Tomyeh

Websocket Channel *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Integration/Miscellaneous/Websocket_Channel *Contributors:* Elenalin, Hawk, Jumperchen, Robertwenzel

Performance Tips *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Performance_Tips *Contributors:* Char, Hawk, Tmillsclare, Tomyeh

Use Compiled Java Codes *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Performance_Tips/Use_Compiled_Java_Codes *Contributors:* Alicelin, Char, Hawk, Jumperchen, Maya001122, RebeccaLai, Tomyeh, Vincent

Use Native Namespace instead of XHTML Namespace *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Performance_Tips/Use_Native_Namespace_instead_of_XHTML_Namespace *Contributors:* Alicelin, Hawk, Jumperchen, RebeccaLai, Tomyeh, Vincent

Use ZK JSP Tags instead of ZK Filter *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Performance_Tips/Use_ZK_JSP_Tags_instead_of_ZK_Filter *Contributors:* Hawk, Maya001122, Tomyeh, Vincent

Defer the Creation of Child Components *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Performance_Tips/Defer_the_Creation_of_Child_Components *Contributors:* Alicelin, Char, Hawk, Maya001122, RebeccaLai, Tomyeh

Defer the Rendering of Client Widgets *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Performance_Tips/Defer_the_Rendering_of_Client_Widgets *Contributors:* Alicelin, Jerrychen, Jumperchen, Maya001122, RebeccaLai, Southerncrossie, Tomyeh, Vincent

Client Render on Demand *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Performance_Tips/Client_Render_on_Demand *Contributors:* Char, Charlesqiu, Hawk, Maya001122, Neillee2, RebeccaLai, Tomyeh

Listbox, Grid and Tree for Huge Data *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Performance_Tips/Performance_Tips/Listbox%2C_Grid_and_Tree_for_Huge_Data *Contributors:* Alicelin, RebeccaLai, Tomyeh

Use Live Data and Paging *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Performance_Tips/Listbox%2C_Grid_and_Tree_for_Huge_Data/Use_Live_Data_and_Paging *Contributors:* Alicelin, Char, Hawk, Maya001122, Sphota, Tomyeh

Turn on Render on Demand *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Performance_Tips/Listbox%2C_Grid_and_Tree_for_Huge_Data/Turn_on_Render_on_Demand *Contributors:* Alicelin, Ashishd, Char, Hawk, Jimmyshiau, Maya001122, ProtonChang, RebeccaLai, SimonPai, Tomyeh

Implement ListModel and TreeModel *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Performance_Tips/Load_JavaScript_and_CSS_from_Server_Nearby *Contributors:* Alicelin, Hawk, Jumperchen, RebeccaLai, Simbo, Tomyeh

Minimize Number of JavaScript Files to Load *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Performance_Tips/Minimize_Number_of_JavaScript_Files_to_Load *Contributors:* Alicelin, Hawk, Jimmyshiau, Jumperchen, RebeccaLai, Tomyeh, Tonyq

Load JavaScript and CSS from Server Nearby *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Performance_Tips/Load_JavaScript_and_CSS_from_Server_Nearby *Contributors:* Alicelin, Char, Flyworld, Hawk, Jumperchen, RebeccaLai, Tomyeh

Specify Stubonly for Client-only Components *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Performance_Tips/Specify_Stubonly_for_Client-only_Components *Contributors:* Alicelin, Char, Chunfuchang, Hawk, Jumperchen, RebeccaLai, Tomyeh, Vincent

Reuse Desktops *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Performance_Tips/Reuse/Desktops *Contributors:* Char, Hawk, Maya001122, Raymondchao, RebeccaLai, Tomyeh

Control resource caching *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Performance_Tips/Control_resource_caching *Contributors:* Hawk, Matthieu, RebeccaLai

Miscellaneous *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Performance_Tips/Miscellaneous *Contributors:* Hawk, Maya001122, RebeccaLai, Tomyeh, Vincent

Security Tips *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Security_Tips *Contributors:* Ashishd, Jeanher, Tomyeh

Cross-site scripting *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Security_Tips/Cross-site_scripting *Contributors:* Alicelin, Hawk, Jameschu, Jeanher, RebeccaLai, Robertwenzel, Tomyeh

Block Request for Inaccessible Widgets *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Security_Tips/Block_Request_for_Inaccessible_Widgets *Contributors:* Alicelin, Dennischen, Hawk, Jameschu, Jeanher, RebeccaLai, Tomyeh

Denial Of Service *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Security_Tips/Denial_Of_Service *Contributors:* Ashishd, Hawk, RebeccaLai

Cross-site Request Forgery *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Security_Tips/Cross-site_Request_Forgery *Contributors:* Ashishd, Hawk, RebeccaLai

OWASP Top 10 Security Concerns In 2017 *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Security_Tips/OWASP_Top_10_Security_Concerns_In_2017 *Contributors:* Hawk, Matthieu, Robertwenzel

Content Security Policy *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Security_Tips/Content_Security_Policy *Contributors:* Hawk, Jameschu, Jeanher, RebeccaLai

SSO Redirect Handling *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Security_Tips/SSO_Redirect_Handling *Contributors:* Hawk, Jeanher, RebeccaLai

Performance Monitoring *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Performance_Monitoring *Contributors:* Alicelin, Hawk, Tomyeh

Performance Meters *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Performance_Monitoring/Performance_Meters *Contributors:* Alicelin, Hawk, RebeccaLai, Tomyeh

Event Interceptors *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Performance_Monitoring/Event_Interceptors *Contributors:* Hawk, Tomyeh

Loading Monitors *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Performance_Monitoring>Loading_Monitors *Contributors:* Hawk, Tomyeh

Step by Step Trouble Shooting *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Performance_Monitoring/Step_by_Step_Trouble_Shooting *Contributors:* Hawk, Phoebelin, RebeccaLai, Robertwenzel, Southerncrossie, Vincent

Accessibility *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Accessibility *Contributors:* Hawk, Jeanher, RebeccaLai, Rudyhuang

Keyboard Support *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Accessibility/Keyboard_Support *Contributors:* Hawk, Jeanher

High Contrast Theme *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Accessibility/High_Contrast_Theme *Contributors:* Hawk, Jeanher

Testing *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Testing *Contributors:* Alicelin, Char, Hawk, RebeccaLai, Tomyeh

Testing Tips *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Testing/Testing_Tips *Contributors:* Alicelin, Char, Hawk, RebeccaLai, Tomyeh, Vincent

ZATS *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Testing/ZATS *Contributors:* Hawk, RebeccaLai, Southerncrossie

Upgrade Tips *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Upgrade_Tips *Contributors:* Jeanher

Version Upgrade *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Upgrade_Tips/Version_Upgrade *Contributors:* Hawk, Jeanher

Edition Upgrade *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Upgrade_Tips/Edition_Upgrade *Contributors:* Jeanher

Customization *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Customization *Contributors:* Tomyeh

Packing Code *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Customization/Packing_Code *Contributors:* Alicelin, Hawk, RebeccaLai, Tomyeh

Component Properties *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Customization/Component_Properties *Contributors:* Alicelin, Hawk, Tomyeh

UI Factory *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Customization/UI_Factory *Contributors:* Alicelin, Hawk, RebeccaLai, Tomyeh

Life Cycle Listener *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Customization/Life_Cycle_Listener *Contributors:* Hawk, RebeccaLai, Tomyeh

AU Services *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Customization/AU_Services *Contributors:* Hawk, Tomyeh

AU Extensions *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Customization/AU_Extensions *Contributors:* Hawk, Tomyeh

How to Build ZK Source Code *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Customization/How_to_Build_ZK_Source_Code *Contributors:* Alicelin, Hawk, Jumperchen, RebeccaLai, Tomyeh

Handle AU Request Resend *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Customization/Handle_AU_Request_Resend *Contributors:* Hawk, Jumperchen, RebeccaLai

Supporting Utilities *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Supporting_Utils *Contributors:* Alicelin, RebeccaLai, Tomyeh

Logger *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Supporting_Utils/Logger *Contributors:* Hawk, Jumperchen, RebeccaLai, Rudyhuang, Tomyeh

DSP *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Supporting_Utils/DSP *Contributors:* Flyworld, Hawk, Tomyeh

iDOM *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Supporting_Utils/iDOM *Contributors:* Hawk, Tomyeh

Common Error Messages *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Common_Error_Messages *Contributors:* Hawk, RebeccaLai

Development-time Best Practices *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Development-time_Best_Practices *Contributors:* Hawk, Robertwenzel

Packaging Applications *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/Packaging_Applications *Contributors:* Hawk, RebeccaLai

How to get Efficient Support *Source:* http://10.1.3.180/index.php?title=ZK_Developer%27s_Reference/How_to_get_Efficient_Support *Contributors:* Hawk, Jeanher

Image Sources, Licenses and Contributors

Image:zk-web-tier.jpg Source: http://10.1.3.180/index.php?title=File:Zk-web-tier.jpg License: unknown Contributors: Hawk

File:Load-page.jpg Source: http://10.1.3.180/index.php?title=File:Load-page.jpg License: unknown Contributors: Hawk

Image:architecture-s.png Source: http://10.1.3.180/index.php?title=File:Architecture-s.png License: unknown Contributors: Tomyeh

Image:ZKEssentials_Intro_Hello.png Source: http://10.1.3.180/index.php?title=File:ZKEssentials_Intro_Hello.png License: unknown Contributors: Sphota

Image:ZKEssentials_Intro_MultiPage.png Source: http://10.1.3.180/index.php?title=File:ZKEssentials_Intro_MultiPage.png License: unknown Contributors: Sphota

Image:zk the id space.jpg Source: http://10.1.3.180/index.php?title=File:Zk_the_id_space.jpg License: unknown Contributors: Maya001122

File:Eventqueue-concept.jpg Source: http://10.1.3.180/index.php?title=File:Eventqueue-concept.jpg License: unknown Contributors: Tomyeh

Image:version_ee.png Source: http://10.1.3.180/index.php?title=File:Version_ee.png License: unknown Contributors: Tmillsclare

Image:version_ce-pe-ee.png Source: http://10.1.3.180/index.php?title=File:Version_ce-pe-ee.png License: unknown Contributors: Tmillsclare

File:MVC.png Source: http://10.1.3.180/index.php?title=File:MVC.png License: unknown Contributors: Tomyeh

Image:Composer.PNG Source: http://10.1.3.180/index.php?title=File:Composer.PNG License: unknown Contributors: Tomyeh

File:model-driven-rendering.jpg Source: http://10.1.3.180/index.php?title=File:Model-driven-rendering.jpg License: unknown Contributors: Hawk

Image:DrListModelRenderer.png Source: http://10.1.3.180/index.php?title=File:DrListModelRenderer.png License: unknown Contributors: Robertwenzel, Tomyeh

Image:version_pe-ee.png Source: http://10.1.3.180/index.php?title=File:Version_pe-ee.png License: unknown Contributors: Tmillsclare

Image:DrGroupsModel.png Source: http://10.1.3.180/index.php?title=File:DrGroupsModel.png License: unknown Contributors: Tomyeh

Image:Grouping_model_explain.png Source: http://10.1.3.180/index.php?title=File:Grouping_model_explain.png License: unknown Contributors: Tomyeh

Image:DrGroupsModelArray.png Source: http://10.1.3.180/index.php?title=File:DrGroupsModelArray.png License: unknown Contributors: Tomyeh

Image:DrTreeModel2.png Source: http://10.1.3.180/index.php?title=File:DrTreeModel2.png License: unknown Contributors: Tomyeh

Image:DrTreeModel1.png Source: http://10.1.3.180/index.php?title=File:DrTreeModel1.png License: unknown Contributors: Tomyeh

File:leafBranch.jpg Source: http://10.1.3.180/index.php?title=File:LeafBranch.jpg License: unknown Contributors: Hawk

File:St201107-listbox.png Source: http://10.1.3.180/index.php?title=File:St201107-listbox.png License: unknown Contributors: Tomyeh

File:St201107-listbox-in-listbox.png Source: http://10.1.3.180/index.php?title=File:St201107-listbox-in-listbox.png License: unknown Contributors: Tomyeh

File:St201311-tabbox.png Source: http://10.1.3.180/index.php?title=File:St201311-tabbox.png License: unknown Contributors: Jumperchen

File:Zk10_compare_client-mvvm.png Source: http://10.1.3.180/index.php?title=File:Zk10_compare_client-mvvm.png License: unknown Contributors: Matthieu

File:Zk10_compare_stateless.png Source: http://10.1.3.180/index.php?title=File:Zk10_compare_stateless.png License: unknown Contributors: Matthieu

File:Fluid_Layouts.PNG Source: http://10.1.3.180/index.php?title=File:Fluid_Layouts.PNG License: unknown Contributors: Jumperchen

File:Adaptive_Layouts.PNG Source: http://10.1.3.180/index.php?title=File:Adaptive_Layouts.PNG License: unknown Contributors: Jumperchen

File:Responsive_Design.png Source: http://10.1.3.180/index.php?title=File:Responsive_Design.png License: unknown Contributors: Jumperchen

File:DrMessagebox.png Source: http://10.1.3.180/index.php?title=File:DrMessagebox.png License: unknown Contributors: Tomyeh

File:DrMessagebox-error.png Source: http://10.1.3.180/index.php?title=File:DrMessagebox-error.png License: unknown Contributors: Tomyeh

Image:DrHLayout.png Source: http://10.1.3.180/index.php?title=File:DrHLayout.png License: unknown Contributors: Tomyeh

Image:DrVLayout.png Source: http://10.1.3.180/index.php?title=File:DrVLayout.png License: unknown Contributors: Tomyeh

Image:DrHLayout_scrolling.png Source: http://10.1.3.180/index.php?title=File:DrHLayout_scrolling.png License: unknown Contributors: Alicelin, Jimmyshiau

Image:DrVLayout_scrolling.png Source: http://10.1.3.180/index.php?title=File:DrVLayout_scrolling.png License: unknown Contributors: Alicelin, Jimmyshiau

Image:DrHLayout_alignment.png Source: http://10.1.3.180/index.php?title=File:DrHLayout_alignment.png License: unknown Contributors: Alicelin, Jimmyshiau

Image:DrHbox.png Source: http://10.1.3.180/index.php?title=File:DrHbox.png License: unknown Contributors: Tomyeh

Image:DrVbox.png Source: http://10.1.3.180/index.php?title=File:DrVbox.png License: unknown Contributors: Tomyeh

Image:DrHbox_align.png Source: http://10.1.3.180/index.php?title=File:DrHbox_align.png License: unknown Contributors: Alicelin, Jimmyshiau

Image:DrHbox_pack.png Source: http://10.1.3.180/index.php?title=File:DrHbox_pack.png License: unknown Contributors: Alicelin, Jimmyshiau

Image:DrVbox_align.png Source: http://10.1.3.180/index.php?title=File:DrVbox_align.png License: unknown Contributors: Alicelin, Jimmyshiau

Image:DrVbox_pack.png Source: http://10.1.3.180/index.php?title=File:DrVbox_pack.png License: unknown Contributors: Alicelin, Jimmyshiau

Image:DrHbox_Cell.png Source: http://10.1.3.180/index.php?title=File:DrHbox_Cell.png License: unknown Contributors: Alicelin, Jimmyshiau

Image:DrVbox_Cell.png Source: http://10.1.3.180/index.php?title=File:DrVbox_Cell.png License: unknown Contributors: Alicelin, Jimmyshiau

Image:DrBorderlayout.png Source: http://10.1.3.180/index.php?title=File:DrBorderlayout.png License: unknown Contributors: Alicelin, Jimmyshiau, Tomyeh

Image:DrBorderlayout_flex.png Source: http://10.1.3.180/index.php?title=File:DrBorderlayout_flex.png License: unknown Contributors: Alicelin, Jimmyshiau

Image:DrBorderlayout_Center_scrolling.png Source: http://10.1.3.180/index.php?title=File:DrBorderlayout_Center_scrolling.png License: unknown Contributors: Alicelin, Jimmyshiau

Image:DrBorderlayout_grow.png Source: http://10.1.3.180/index.php?title=File:DrBorderlayout_grow.png License: unknown Contributors: Alicelin, Jimmyshiau

Image:DrColumnlayout.png Source: http://10.1.3.180/index.php?title=File:DrColumnlayout.png License: unknown Contributors: Tomyeh

Image:DrPortallayout.png Source: http://10.1.3.180/index.php?title=File:DrPortallayout.png License: unknown Contributors: Tomyeh

Image:DrTablelayout.png Source: http://10.1.3.180/index.php?title=File:DrTablelayout.png License: unknown Contributors: Tomyeh

Image:DrDivSpan.png Source: http://10.1.3.180/index.php?title=File:DrDivSpan.png License: unknown Contributors: Tomyeh

Image:DrDiv_scrolling.png Source: http://10.1.3.180/index.php?title=File:DrDiv_scrolling.png License: unknown Contributors: Alicelin, Jimmyshiau

Image:DrWindow.png Source: http://10.1.3.180/index.php?title=File:DrWindow.png License: unknown Contributors: Tomyeh

Image:DrWindow_scrolling.png Source: http://10.1.3.180/index.php?title=File:DrWindow_scrolling.png License: unknown Contributors: Alicelin, Jimmyshiau

Image:DrPanel.png Source: http://10.1.3.180/index.php?title=File:DrPanel.png License: unknown Contributors: Tomyeh

Image:DrPanel_scrolling.png Source: http://10.1.3.180/index.php?title=File:DrPanel_scrolling.png License: unknown Contributors: Alicelin, Jimmyshiau

Image:DrGroupBox3d.png Source: http://10.1.3.180/index.php?title=File:DrGroupBox3d.png License: unknown Contributors: Tomyeh

Image:DrGroupBox3d_scrolling.png Source: http://10.1.3.180/index.php?title=File:DrGroupBox3d_scrolling.png License: unknown Contributors: Jimmyshiau

Image:DrTabbox.png Source: http://10.1.3.180/index.php?title=File:DrTabbox.png License: unknown Contributors: Tomyeh

Image:DrTabbox_scrolling.png Source: http://10.1.3.180/index.php?title=File:DrTabbox_scrolling.png License: unknown Contributors: Alicelin, Jimmyshiau

Image:DrFlex1.png Source: http://10.1.3.180/index.php?title=File:DrFlex1.png License: unknown Contributors: Tomyeh

Image:DrFlexTabbox.png Source: http://10.1.3.180/index.php?title=File:DrFlexTabbox.png License: unknown Contributors: Tomyeh

Image:DrFlex2.png Source: http://10.1.3.180/index.php?title=File:DrFlex2.png License: unknown Contributors: Tomyeh

Image:DrFlexErr1Fix.png Source: http://10.1.3.180/index.php?title=File:DrFlexErr1Fix.png License: unknown Contributors: Tomyeh

Image:flexborderlayout.png Source: http://10.1.3.180/index.php?title=File:flexborderlayout.png License: unknown Contributors: Elton776, Jimmyshiau

File:ZK6DevRef_Vlayout_Hflex.png Source: http://10.1.3.180/index.php?title=File:ZK6DevRef_Vlayout_Hflex.png License: unknown Contributors: Vincent

File:ZK6DevRef_Vlayout_Vflex.png Source: http://10.1.3.180/index.php?title=File:ZK6DevRef_Vlayout_Vflex.png License: unknown Contributors: Vincent

Image:DrGridFlex.png Source: http://10.1.3.180/index.php?title=File:DrGridFlex.png License: unknown Contributors: Tomyeh

File:ZK5DevRef_GridColumn_FormHflex.png Source: http://10.1.3.180/index.php?title=File:ZK5DevRef_GridColumn_FormHflex.png License: unknown Contributors: Alicelin, Jimmyshiau

File:ZK5DevRef_GridColumn_FormHflex2.png Source: http://10.1.3.180/index.php?title=File:ZK5DevRef_GridColumn_FormHflex2.png License: unknown Contributors: Alicelin, Jimmyshiau

File:ZK5DevRef_GridColumn_FormHflex_colspan.png Source: http://10.1.3.180/index.php?title=File:ZK5DevRef_GridColumn_FormHflex_colspan.png License: unknown Contributors: Alicelin, Jimmyshiau

Image:DrFlexErr1.png Source: http://10.1.3.180/index.php?title=File:DrFlexErr1.png License: unknown Contributors: Tomyeh

Image:DrFlexErr2.png Source: http://10.1.3.180/index.php?title=File:DrFlexErr2.png License: unknown Contributors: Tomyeh

File:ZK5DevRef_GridColumn_Default.png Source: http://10.1.3.180/index.php?title=File:ZK5DevRef_GridColumn_Default.png License: unknown Contributors: Flyworld

File:ZK5DevRef_GridColumn_hflex.png Source: http://10.1.3.180/index.php?title=File:ZK5DevRef_GridColumn_hflex.png License: unknown Contributors: Flyworld

File:ZK5DevRef_GridColumn_nospan.png Source: http://10.1.3.180/index.php?title=File:ZK5DevRef_GridColumn_nospan.png License: unknown Contributors: Flyworld

File:ZK5DevRef_GridColumn_span.png Source: http://10.1.3.180/index.php?title=File:ZK5DevRef_GridColumn_span.png License: unknown Contributors: Flyworld

File:ZK5DevRef_GridColumn_span0.png Source: http://10.1.3.180/index.php?title=File:ZK5DevRef_GridColumn_span0.png License: unknown Contributors: SimonPai

File:ZK5DevRef_GridColumn_sizedByCnt.png Source: http://10.1.3.180/index.php?title=File:ZK5DevRef_GridColumn_sizedByCnt.png License: unknown Contributors: SimonPai

File:ZK5DevRef_GridColumn_DefaultLong.png Source: http://10.1.3.180/index.php?title=File:ZK5DevRef_GridColumn_DefaultLong.png License: unknown Contributors: Flyworld

File:ZK5DevRef_GridColumn_DefaultWidth.png Source: http://10.1.3.180/index.php?title=File:ZK5DevRef_GridColumn_DefaultWidth.png License: unknown Contributors: Flyworld

File:ZK5DevRef_GridColumn_LongHflex.png Source: http://10.1.3.180/index.php?title=File:ZK5DevRef_GridColumn_LongHflex.png License: unknown Contributors: Flyworld

File:ZK5DevRef_GridColumn_MixhflexnumWidth.png Source: http://10.1.3.180/index.php?title=File:ZK5DevRef_GridColumn_MixhflexnumWidth.png License: unknown Contributors: Flyworld

File:ZK5DevRef_GridColumn_MixhflexMinWidth.png Source: http://10.1.3.180/index.php?title=File:ZK5DevRef_GridColumn_MixhflexMinWidth.png License: unknown Contributors: Flyworld

File:ZK5DevRef_GridColumn_MixhflAllh.png Source: http://10.1.3.180/index.php?title=File:ZK5DevRef_GridColumn_MixhflAllh.png License: unknown Contributors: Flyworld, Jimmyshiau

Image:DrTooltip.png Source: http://10.1.3.180/index.php?title=File:DrTooltip.png License: unknown Contributors: Tomyeh

Image:drContext.png Source: http://10.1.3.180/index.php?title=File:DrContext.png License: unknown Contributors: Tomyeh

Image:100000000000017500000052E60F488A.png Source: http://10.1.3.180/index.php?title=File:100000000000017500000052E60F488A.png License: unknown Contributors: Maya001122

File:events_1_finger.jpg Source: http://10.1.3.180/index.php?title=File:Events_1_finger.jpg License: unknown Contributors: Jimmyshiau

File:OnBeforeUnload1.png Source: http://10.1.3.180/index.php?title=File:OnBeforeUnload1.png License: unknown Contributors: Matthieu

File:OnBeforeUnload2.png Source: http://10.1.3.180/index.php?title=File:OnBeforeUnload2.png License: unknown Contributors: Matthieu

Image:10000000000002AF000001BB582C2DD7.png Source: http://10.1.3.180/index.php?title=File:10000000000002AF000001BB582C2DD7.png License: unknown Contributors: Char

Image:confirmClose.png Source: http://10.1.3.180/index.php?title=File:ConfirmClose.png License: unknown Contributors: Hawk

Image:1000000000000284000000226A7DDEE65.png Source: http://10.1.3.180/index.php?title=File:1000000000000284000000226A7DDEE65.png License: unknown Contributors: Maya001122

File:DrSessTimeout.png Source: http://10.1.3.180/index.php?title=File:DrSessTimeout.png License: unknown Contributors: Tomyeh

Image:Exception.png Source: http://10.1.3.180/index.php?title=File:Exception.png License: unknown Contributors: Tomyeh

File:error_handling_crash_screen.png Source: http://10.1.3.180/index.php?title=File:Error_handling_crash_screen.png License: unknown Contributors: Robertwenzel

Image:Exception-au.png Source: http://10.1.3.180/index.php?title=File:Exception-au.png License: unknown Contributors: Tomyeh

Image:Exception-au2.png Source: http://10.1.3.180/index.php?title=File:Exception-au2.png License: unknown Contributors: Tomyeh

Image:ZKDevRef_UIPattern_UsefulJavaUtil_Clients_showBusy.png Source: http://10.1.3.180/index.php?title=File:ZKDevRef_UIPattern_UsefulJavaUtil_Clients_showBusy.png License: unknown Contributors: SimonPai

Image:ZKDevRef_UIPattern_UsefulJavaUtil_Clients_showNotification01.png Source: http://10.1.3.180/index.php?title=File:ZKDevRef_UIPattern_UsefulJavaUtil_Clients_showNotification01.png License: unknown Contributors: SimonPai

Image:ZKDevRef_UIPattern_UsefulJavaUtil_Clients_showNotification02.png Source: http://10.1.3.180/index.php?title=File:ZKDevRef_UIPattern_UsefulJavaUtil_Clients_showNotification02.png License: unknown Contributors: SimonPai

Image:ZKDevRef_UIPattern_UsefulJavaUtil_Clients_showNotification03.png Source: http://10.1.3.180/index.php?title=File:ZKDevRef_UIPattern_UsefulJavaUtil_Clients_showNotification03.png License: unknown Contributors: SimonPai

Image:ZKDevRef_UIPattern_UsefulJavaUtil_Clients_showNotification04.png Source: http://10.1.3.180/index.php?title=File:ZKDevRef_UIPattern_UsefulJavaUtil_Clients_showNotification04.png License: unknown Contributors: Vincent

Image:ZKDevRef_UIPattern_UsefulJavaUtil_Toast01.png Source: http://10.1.3.180/index.php?title=File:ZKDevRef_UIPattern_UsefulJavaUtil_Toast01.png License: unknown Contributors: Rudyhuang

File:Loadingbar.gif Source: http://10.1.3.180/index.php?title=File:Loadingbar.gif License: unknown Contributors: Jeanher, Leonlee

File:Loadingbar2.gif Source: http://10.1.3.180/index.php?title=File:Loadingbar2.gif License: unknown Contributors: Jeanher

Image:html_1.png Source: http://10.1.3.180/index.php?title=File:Html_1.png License: unknown Contributors: Char

Image:XML_SVG.png Source: http://10.1.3.180/index.php?title=File:XML_SVG.png License: unknown Contributors: Char

File:Icon_info.png Source: http://10.1.3.180/index.php?title=File:Icon_info.png License: unknown Contributors: Hawk, Tmillsclare

File:breeze-look-and-feel.png Source: http://10.1.3.180/index.php?title=File:Breeze-look-and-feel.png License: unknown Contributors: Vincent

File:sapphire-look-and-feel.png Source: http://10.1.3.180/index.php?title=File:Sapphire-look-and-feel.png License: unknown Contributors: Vincent

File:silvertail-look-and-feel.png Source: http://10.1.3.180/index.php?title=File:Silvertail-look-and-feel.png License: unknown Contributors: Vincent

File:atlantic-look-and-feel.png Source: http://10.1.3.180/index.php?title=File:Atlantic-look-and-feel.png License: unknown Contributors: Vincent

File:ZK85_Theme_Lite.png Source: http://10.1.3.180/index.php?title=File:ZK85_Theme_Lite.png License: unknown Contributors: Michellechen

File:ZK85_Theme_Dark.png Source: http://10.1.3.180/index.php?title=File:ZK85_Theme_Dark.png License: unknown Contributors: Michellechen

File:ZK85_Theme_Mix.png Source: http://10.1.3.180/index.php?title=File:ZK85_Theme_Mix.png License: unknown Contributors: Michellechen

File:Wcag_blue.png Source: http://10.1.3.180/index.php?title=File:Wcag_blue.png License: unknown Contributors: Hawk

File:Wcag_navy.png Source: http://10.1.3.180/index.php?title=File:Wcag_navy.png License: unknown Contributors: Hawk

File:Wcag_purple.png Source: http://10.1.3.180/index.php?title=File:Wcag_purple.png License: unknown Contributors: Hawk

Image:stop.png Source: http://10.1.3.180/index.php?title=File:Stop.png License: unknown Contributors: Tmillsclare

File:theme_skeleton.png Source: http://10.1.3.180/index.php?title=File:Theme_skeleton.png License: unknown Contributors: Neillee

File:ThemeSubsystem.png Source: http://10.1.3.180/index.php?title=File:ThemeSubsystem.png License: unknown Contributors: Neillee2

File:Universal_time_conversion.png Source: http://10.1.3.180/index.php?title=File:Universal_time_conversion.png License: unknown Contributors: Matthieu

Image:DrForm.png Source: http://10.1.3.180/index.php?title=File:DrForm.png License: unknown Contributors: Tomyeh

Image:html_5.png Source: http://10.1.3.180/index.php?title=File:Html_5.png License: unknown Contributors: Char

File:Hibernate-beginning.png Source: http://10.1.3.180/index.php?title=File:Hibernate-beginning.png License: unknown Contributors: Hawk

File:spring-security-anonymous.png Source: http://10.1.3.180/index.php?title=File:Spring-security-anonymous.png License: unknown Contributors: Hawk

File:spring-security-user.png Source: http://10.1.3.180/index.php?title=File:Spring-security-user.png License: unknown Contributors: Hawk

File:spring-security-editor.png Source: http://10.1.3.180/index.php?title=File:Spring-security-editor.png License: unknown Contributors: Hawk

File:Rod_demonstration.gif Source: http://10.1.3.180/index.php?title=File:Rod_demonstration.gif License: unknown Contributors: Hawk

File:URLEncoder.png Source: http://10.1.3.180/index.php?title=File:URLEncoder.png License: unknown Contributors: Flyworld

File:Redirect302.jpg Source: http://10.1.3.180/index.php?title=File:Redirect302.jpg License: unknown Contributors: Hawk

Image:performancemeter.png Source: http://10.1.3.180/index.php?title=File:Performancemeter.png License: unknown Contributors: Elton776

File:performance_debug.png Source: http://10.1.3.180/index.php?title=File:Performance_debug.png License: unknown Contributors: Hawk

File:chrome_developer_tools_network.png Source: http://10.1.3.180/index.php?title=File:Chrome_developer_tools_network.png License: unknown Contributors: Robertwenzel

File:chrome_developer_tools_network_timing.png Source: http://10.1.3.180/index.php?title=File:Chrome_developer_tools_network_timing.png License: unknown Contributors: Hawk, Robertwenzel

File:js_profile_flame_chart.png Source: http://10.1.3.180/index.php?title=File:Js_profile_flame_chart.png License: unknown Contributors: Robertwenzel

File:js_timeline_events.png Source: http://10.1.3.180/index.php?title=File:Js_timeline_events.png License: unknown Contributors: Robertwenzel

File:suspended_process.png Source: http://10.1.3.180/index.php?title=File:Suspended_process.png License: unknown Contributors: Robertwenzel

File: sampler-result.png Source: http://10.1.3.180/index.php?title=File:Sampler-result.png License: unknown Contributors: Robertwenzel

File: sampler-snapshot-combined.png Source: http://10.1.3.180/index.php?title=File:Sampler-snapshot-combined.png License: unknown Contributors: Robertwenzel

File:heap-dump.png Source: http://10.1.3.180/index.php?title=File:Heap-dump.png License: unknown Contributors: Robertwenzel

File:heap-dump-instances.png Source: http://10.1.3.180/index.php?title=File:Heap-dump-instances.png License: unknown Contributors: Robertwenzel

File:profile-only-package.jpg Source: http://10.1.3.180/index.php?title=File:Profile-only-package.jpg License: unknown Contributors: Hawk

File: hotSpots.jpg Source: http://10.1.3.180/index.php?title=File:HotSpots.jpg License: unknown Contributors: Hawk

File: forwardCall.jpg Source: http://10.1.3.180/index.php?title=File:ForwardCall.jpg License: unknown Contributors: Hawk

File: Wcag_demo.png Source: http://10.1.3.180/index.php?title=File:Wcag_demo.png License: unknown Contributors: Jeanher

File: wcag_themes.png Source: http://10.1.3.180/index.php?title=File:Wcag_themes.png License: unknown Contributors: Hawk

File:Smalltalk-ZatsMimicConcept.png Source: http://10.1.3.180/index.php?title=File:Smalltalk-ZatsMimicConcept.png License: unknown Contributors: Hawk

File:Smalltalk-mimic-hello.png Source: http://10.1.3.180/index.php?title=File:Smalltalk-mimic-hello.png License: unknown Contributors: Hawk

File:no-longer-available.png Source: http://10.1.3.180/index.php?title=File:No-longer-available.png License: unknown Contributors: Hawk

File:HierachicalClassLoader.jpg Source: http://10.1.3.180/index.php?title=File:HierachicalClassLoader.jpg License: unknown Contributors: Hawk

File:ZK_MeetRproblem01.png Source: http://10.1.3.180/index.php?title=File:ZK_MeetRproblem01.png License: unknown Contributors: Jimmyshiau

File:chrome-js-error.png Source: http://10.1.3.180/index.php?title=File:Chrome_js-error.png License: unknown Contributors: Hawk

File:save-har-firefox.png Source: http://10.1.3.180/index.php?title=File:Save-har-firefox.png License: unknown Contributors: Hawk

File:au-request.png Source: http://10.1.3.180/index.php?title=File:Au-request.png License: unknown Contributors: Hawk

File:jettyRoot.png Source: http://10.1.3.180/index.php?title=File:JettyRoot.png License: unknown Contributors: Hawk

File:listFolder.png Source: http://10.1.3.180/index.php?title=File>ListFolder.png License: unknown Contributors: Hawk