The CareWeb Framework

Programmer Guide

Version 3.0

13 June 2014

Regenstrief Center for Biomedical Informatics
http://www.carewebframework.org

Table of Contents

Introduction

The CareWeb Framework (CWF) enables the software developer to build complex, highly interactive, web-based applications in a modular fashion. While it was originally conceived to facilitate the creation of web applications in the healthcare space, the CWF itself is domain agnostic and can be used as the foundation for other types of applications as well.

In addition to offering an extensible component model for application development, the CWF provides several important services:

- Component registration and discovery permits a component to declare its presence to the CWF and be discovered by other components.
- Event management provides a simple, yet powerful, publish/subscribe event model and the capability of delivering events locally within an application instance, or remotely to subscribers that may reside almost anywhere.
- Context management permits collaborative sharing of common contexts such as the current patient.
- Layout management allows a user to design, save, and recall a user interface layout.
- The online help subsystem enables a content specialist to produce help content using industry-standard tools and integrate it within the application.

The CWF leverages several open technologies. Foremost among these are:

- Spring Framework – Manages lifecycle, scope, and discovery of shared services and plugins.
- Spring Security – Provides flexible methods for user authentication and authorization.
- ZK Framework – Provides a rich, web-based user interface, but with a programming paradigm that more closely approximates that of thick client development.
- JQuery – A widely used JavaScript library upon which the ZK framework is built.
- Apache Commons Libraries – Provide a number of useful general-purpose functions.
- Apache ActiveMQ Server – A Java Messaging Service (JMS) implementation used for global event delivery.
- Apache Maven – A dependency management system and artifact repository.

Purpose

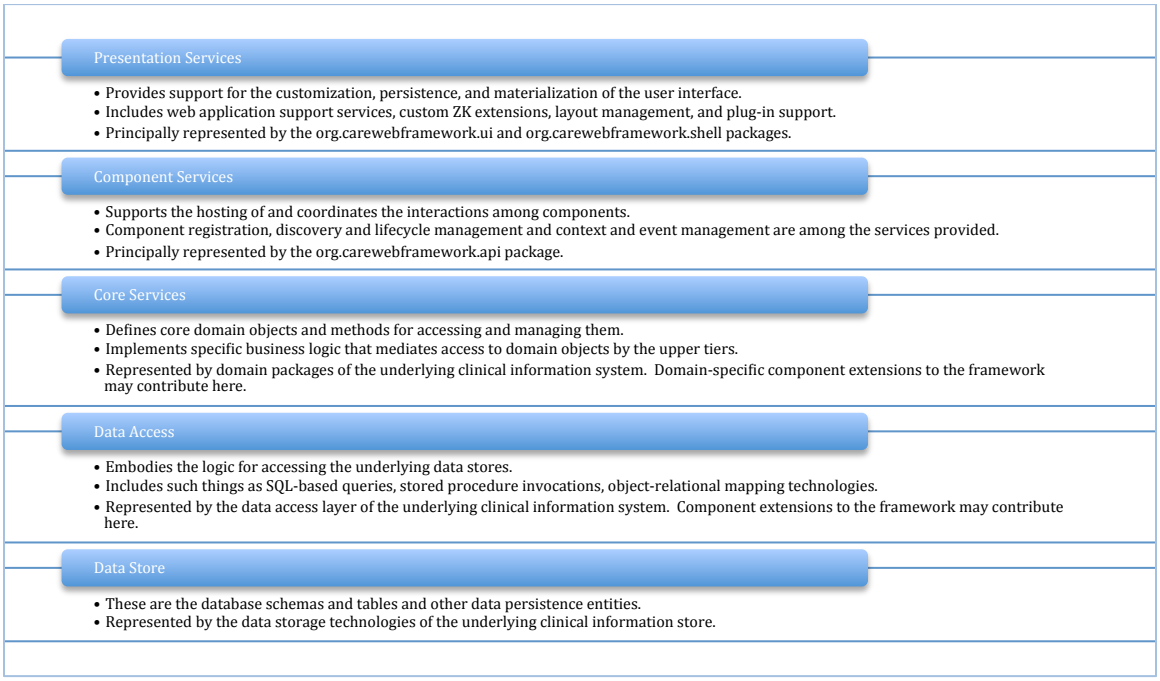
The purpose of this document is to provide the software developer detailed information about the essential aspects of the CWF. It is not meant to exhaustively document every feature, but rather provide what is needed to become proficient in developing applications that leverage and extend the Framework's capabilities.

Conventions

Most API's are documented as tables with the method name on the left and the fully specified package name on the right. A method name tagged with an "s" superscript indicates that it is static while a "c" superscript indicates a constructor. To emphasize code examples as well as method, property, tag, attribute and file names where they occur in narrative text, we use **`code style`**. Method descriptions may list some parameters as optional. This means that there exist alternative method signatures (method overloads) that permit omission of these parameters. Where these occur, we indicate the default value of the omitted parameter. This allows for more succinct documentation of these alternative forms.

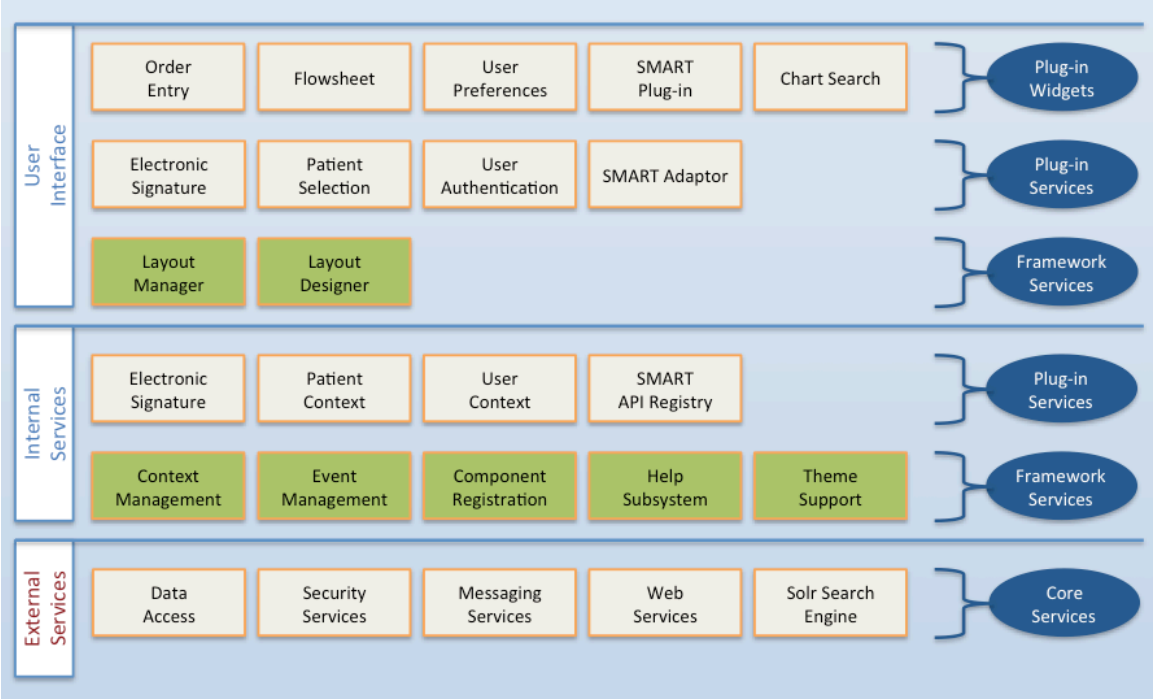
The Architecture

The CWF architecture can best be understood by examining the tiers that underlie it:



The CWF occupies the top two tiers, with domain-specific extensions in the lower tiers. This document will focus on the top two tiers, the component and presentation services, as well as some of the supporting technologies that they leverage.

An alternative view of the CWF from a more functional perspective is shown below. Here, services native to the CWF are highlighted in green while the modular extensions are in white. Internal services are registered to and managed by the CWF while external services are not.



A Brief Spring Framework Tutorial

The CWF makes extensive use of the Spring Framework's Inversion of Control (IOC) container. A detailed description of the Spring Framework is beyond the scope of this document, but may be found at the supporting website. However, a basic understanding of bean management by the Spring Framework is essential in understanding the CWF.

The Spring IOC container is represented by the `ApplicationContext` class and its descendants. The container manages the lifecycle of the objects (beans) that it creates. Configuration files, using an XML-based declarative syntax, control what beans get created and how they are created and can inject property values into those beans. Property values may be constants, references to other managed beans, or Expression Language (EL) style expressions. In addition, containers may be chained together in a hierarchical fashion, with child containers having access to beans declared in their ancestor containers. This hierarchical capability is critical in a web environment such as CWF where it is often necessary to define different scopes for shared resources.

The CWF creates Spring containers in two different scopes, a **root scope** and a **desktop scope**. The root scope spans the entire web application. Beans declared within the root container are created when the web server is started and are accessible to all code running within the web application. Thread safety is an especially important concern when creating beans in the root scope as they may be accessed by multiple client processes concurrently.

In contrast, a desktop-scoped container is created and bound to each new desktop instance (roughly corresponding to a browser window). This container is accessible only to code belonging to that desktop instance. Since the root container is the parent of the desktop container, beans managed within the root scope are available to the desktop scope. The reverse, however, is never true.

The Spring IOC Container model is a powerful way to configure and manage shared resources within an application. It can ensure that shared resources are created at the appropriate time in the appropriate scope and provides means for accessing those resources in a consistent manner. The heart of the container model is the configuration file. Configuration data contained within this file use a declarative, XML-based syntax to define the beans that are to be managed. When a container is created, it references one or more of these configuration files. A key feature of Spring is the ability to dynamically discover configuration files in the environment using Ant-style wildcard paths. This feature is used heavily by the CWF to dynamically discover plugin components. In other words, a plugin uses its configuration file to make itself known to the CWF.

A Spring configuration file follows a simple XML syntax. A sample entry is shown below.

```
<?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:cwf="http://www.carewebframework.org/schema/spring"
   xsi:schemaLocation="
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
      http://www.carewebframework.org/schema/spring
      http://www.carewebframework.org/schema/spring/spring-extensions.xsd">

   <beans profile="root">

      <!-- Allows the application to locate the Spring application context -->
      <bean id="appContextFinder"
        class="org.carewebframework.ui.spring.AppContextFinder"
        cwf:override="always" />

      <!-- Registry for shared icon libraries. -->
      <bean id="iconLibraryRegistry"
        class="org.carewebframework.ui.icons.IconLibraryRegistry" />

   </beans>

   <beans profile="desktop">

      <!-- Manages subscription and publication of generic events. -->
      <bean id="eventManager" class="org.carewebframework.ui.event.EventManager"
        destroy-method="destroy" cwf:override="always">
        <property name="desktop" ref="desktop" />
      </bean>

      <bean id="popupManager" class="org.carewebframework.ui.zk.PopupManager"
        destroy-method="destroy">
      </bean>

   </beans>
</beans>
```

At top level of the XML hierarchy is the **`beans`** tag within which are another set of **`beans`** tags each with an associated **`profile`** name. The profile name corresponds to the container scope to which the bean declarations within will belong. Within each profile are a series of bean declarations using the **`bean`** tag. Each bean declaration has several attributes, some of which are optional. Each bean is usually given a unique identifier (the **`id`** attribute). This identifier may be referenced by other bean declarations as shown here and may also be used to retrieve an instance of a specific bean from its container. While an identifier is not required, in most cases you will want to provide one. It is important to provide one that is unlikely to collide with an existing identifier (unless you intend to override a previous declaration – but more on that later).

The **`class`** attribute of a bean declaration must contain the fully qualified class name for the bean. By default, Spring will use the default constructor to create an instance of the bean. This behavior is easily modified by explicitly declaring constructor

arguments (not shown here). In this case, Spring will use the constructor that matches the number and type of provided constructor arguments. As an alternative, one can use a factory method to create an instance of a bean, designated by the **`factory-method`** attribute. In this case, the class attribute is the name of the class implementing the static factory method and the return type of the method determines the actual class of the created bean.

Other common attributes supported by the bean declaration are the **`init-method`** and **`destroy-method`** attributes. These direct the container to call methods on the bean that are executed after the bean is fully instantiated (and its property values set) and before its container releases the bean, respectively.

Yet another important attribute is the **`scope`** attribute. Do not confuse this with the previous scope discussion. In this context, scope refers to whether or not a bean will be a singleton instance within its container's scope. The default value is **`singleton`**. That is, the container creates only one instance of the bean and every request for that bean will produce the same instance. In contrast, specifying a value of **`prototype`** for the scope attribute will result in a unique instance for every bean request. It is important to recognize that the container does not manage the lifecycle of beans created with a prototype scope beyond the initial instantiation phase. Once created and initialized, the container relinquishes all control of a prototype bean. This means that while one can apply the **`init-method`** attribute to a prototype bean, the **`destroy-method`** attribute has no effect. Use of prototype scope is less common than singleton, but there are many cases where this is what is desired (mostly commonly when declaring stateful controller classes).

There are scopes other than singleton and prototype that are supported by Spring. These scopes, however, are not used by the CWF and will not be discussed here. Custom scopes are also possible. The CWF does make use of this capability, implementing a custom **`desktop`** scope – more on this under the section on Spring extensions.

For beans with a singleton scope, one may specify an additional attribute called **`lazy-init`**. By default, a bean with a singleton scope is created when its container is initialized. By specifying a value of true for **`lazy-init`**, instantiation of the bean is delayed until it is first requested. This, of course, has no affect on a bean with a prototype scope as these are, by necessity, lazily instantiated.

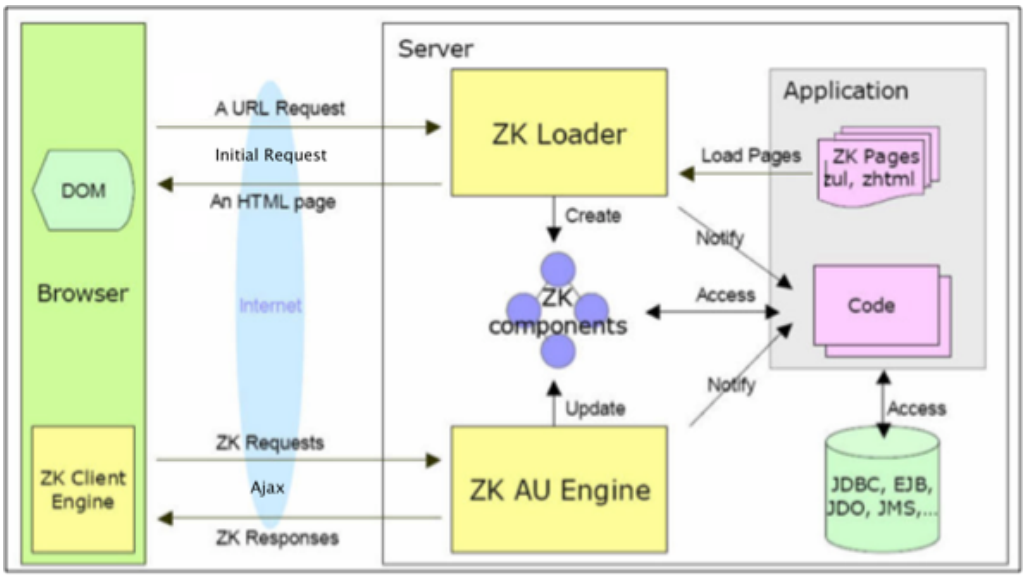
To set property values for a managed bean, use the **`property`** tag. The **`name`** attribute names the property following standard Java bean conventions. To set the value of the property, you may use either the **`value`** attribute to set the value to a constant, or the **`ref`** attribute (shown here) to set the value to that of another managed bean. To set properties that are complex types (lists, arrays, etc.), see the Spring documentation.

There are many variations of the bean declaration syntax that allow you to be very granular in how managed beans are created. Refer to the Spring Framework

documentation for further details. In addition, the CWF has implemented several extensions to provide more granular control over bean overriding and simplify the declaration of plugins and associated resources. These enhancements are discussed in detail later in this document.

A Brief ZK Framework Tutorial

The ZK Framework is a powerful, open-source, Ajax-enabled web application framework that provides a rich user interface experience on par with thick client applications and employs a programming paradigm more closely aligned with thick client development than traditional web application development. ZK effectively blurs the separation between client- and server-side development by creating a unified model of web application development. ZK allows the developer to create and modify the user interface either in server-side code or in file resources (zul pages) using an XML-based mark-up language (ZUML). ZK's Ajax-based user interface engine, server (AU or asynchronous update) and client components working together, transparently synchronizes changes to the server-side model of the user interface with the client and conveys events triggered in the client to event handlers on the server. This architecture is represented below.



Typically, the developer does not need to write any JavaScript, though this capability is on occasion needed and is supported by the ZK Framework. Additionally, most, though not all, browser-specific behaviors and anomalies are shielded from the developer. The ZK Framework also natively integrates with a number of third-party technologies including Spring Framework, JQuery, JFreeChart, Adobe Flash, Groovy, Jython, JSP and many others.

While a detailed treatment of the ZK Framework is well beyond the scope of this document, some basic precepts are presented here. Additional detail is also covered in the section on writing plugins. Fully detailed documentation may be obtained from the ZK supporting website.

The ZK Framework uses a UI component model very similar to traditional thick client models such as Windows Foundation Classes, .NET's Winforms Library or Delphi's Visual Component Library. As with these traditional models, ZK UI

components have parent-child relationships, event handlers, and properties and methods that affect their presentation and behavior. A developer can create a ZK-based UI in one or both of two ways, using the ZUML markup language in a zul page (i.e., web documents ending in a **`.zul`** extension) or programmatically. Regardless of which method is chosen (and almost always it will be a combination of both), ZK creates an in-memory model of the UI on the server and corresponding components (called widgets in ZK parlance) on the client web browser. Changes to component properties on the server are automatically reflected in the client. Events generated at the client (such as clicking on a button) are automatically sent to server-based event handlers.

The ZUML markup language syntax closely mirrors ZK's underlying component model, with each tag having a corresponding component class. Consider the following sample zul page with some simple ZUML markup:

```
<zk>
    <window id="myWindow">
        <label id="aLabel" value="Click the button please."/>
        <button id="aButton" label="Click Me"/>
    </window>
</zk>
```

While the **`zk`** root tag is not required if you have only one top-level element (the window in this example), you should generally use it. It has the added benefit that, by including the ZUML schema declaration (not shown here), you will have syntax checking and code completion capabilities in most development environments. In the above example, we have declared three ZK UI components (or more succinctly, zul components): a window, a label, and a button. The window is the parent component of the other two by virtue of its position in the XML hierarchy. Many zul components have a common set of properties (represented as attributes in ZUML). One such attribute is the **`id`** attribute. This is not required, but is usually given and facilitates referencing the component from code. The **`id`** value must be unique within what ZK calls an "id space". A few zul components define their own id space. Window is one example. Because the zul window component is defined as an id space owner, the ids of its child components must be unique only within that scope. Extending the above example, if we declared a second window component (whether as a child or a sibling of the existing window), the ids of its children would be in a separate id space from the others. Understanding id spaces is important because they define the scope in which a component's identifier lives and can be accessed. Refer to the ZK developer's manual for more detail.

In addition to the **`id`** property, a zul component typically has many properties that may be set using standard attribute syntax in the ZUML declaration. Some properties are specific to a particular component; others are shared by many components. Examples of the latter are the **`visible`** and **`style`** properties. These are shared by most, though not all, zul components. The **`visible`** property controls the visibility of the component (and its children) in the UI. The **`style`** property

permits applying CSS styling directly to the component. For a full inventory of zul components and their properties, refer to the ZK developer's documentation.

Returning to the above sample zul page, one can create an identical UI in code. The in-memory representation would be identical to that created by ZK when it compiles the ZUML declaration. Consider the following Java code snippet:

```
Window myWindow = new Window();
myWindow.setId("myWindow");
Label label = new Label("Click the button please.");
label.setId("aLabel");
label.setParent(myWindow);
Button button = new Button("Click Me");
button.setId("aButton");
button.setParent(myWindow);
```

This code would create the same in-memory model as the zul page above. Note that the ZK Java classes correspond to the ZUML tags by the same name.

The typical programming task in ZK is to create a zul page and define your UI layout there. You can make your zul page dynamic by including such things as EL expressions and ZUML constructs like the **`foreach`** directive (see the ZK documentation). You can declare event handlers explicitly in ZUML by using the corresponding attributes (like **`onClick`**, for example), or you can use the automatic event binding capabilities of the ZK framework. Your event handler code can live inside a zul page as imbedded zscript (a generally discouraged practice for reasons of performance and maintainability) or in external class declarations. You can also bind your zul page to Java code by using either (or both) the traditional MVC approach with a controller class via the **`apply`** attribute or using the traditional thick client approach by subclassing existing zul classes via the **`use`** attribute. More detail on these approaches can be found in the section on writing plugins – both are used extensively throughout the framework.

This only scratches the surface of the capabilities of the ZK Framework. We strongly recommend reviewing the documentation available on the ZK website. There are also several books (see reference section) that provide an introduction to ZK application development.

Organization

The CWF is organized into several physically distinct jars, each performing a related set of functions and each identified by a unique Maven artifact identifier. All framework artifacts share a common Maven group identifier of **`org.carewebframework`**. Below are the Java packages that comprise the Framework, organized under the corresponding Maven artifact identifier.

org.carewebframework.api	
Java Package	**Description**
org.carewebframework.api	This is a set of abstract classes and core features.
org.carewebframework.api.context	Provides management of shared contexts as well as abstract base classes for creating shared contexts.
org.carewebframework.api.domain	Provides classes for deriving common domain objects, such as user.
org.carewebframework.api.event	Provides support for generic events.
org.carewebframework.api.logging	Provides generic logging-related functions.
org.carewebframework.api.property	Classes for abstracting storage and retrieval of persisted property values.
org.carewebframework.api.security	Classes for accessing security services independent of the underlying implementation.
org.carewebframework.api.spring	Extensions to the Spring Framework.
org.carewebframework.api.thread	Provides thread process support.
org.carewebframework.common	
Java Package	**Description**
org.carewebframework.common	This is a collection of classes and convenience methods for manipulating strings, numbers, dates, etc.
org.carewebframework.ext.logging	
Java Package	**Description**
org.carewebframework.logging.log4j	Log4j framework extensions.
org.carewebframework.ext.performance	
Java Package	**Description**
org.carewebframework.logging.perf4j	Perf4j performance logging extensions.
org.carewebframework.help	
Java Package	**Description**
org.carewebframework.help	Help subsystem

org.carewebframework.help.javahelp	
Java Package	**Description**
org.carewebframework.help.javahelp	JavaHelp support for help subsystem.

org.carewebframework.jms.core	
Java Package	**Description**
org.carewebframework.jms	Support of CWF event delivery via JMS-compliant messaging services.

org.carewebframework.jms.activemq	
Java Package	**Description**
org.carewebframework.jms.activemq	Support of CWF event delivery via ActiveMQ JMS implementation.

org.carewebframework.mvn.archetype.plugin	
Java Package	**Description**
none	Maven template for creating UI plugins.

org.carewebframework.mvn.plugin.core	
Java Package	**Description**
org.carewebframework.maven.plugin.core	Custom Maven plugin support.

org.carewebframework.mvn.plugin.helpconverter	
Java Package	**Description**
org.carewebframework.maven.plugin.helpconverter	Maven plugin to convert help content to CWF format.

org.carewebframework.mvn.plugin.themegenerator	
Java Package	**Description**
org.carewebframework.maven.plugin.themegenerator	Maven plugin to generate visual themes.

org.carewebframework.security.spring	
Java Package	**Description**
org.carewebframework.security.spring	Basic support for Spring Security integration.

org.carewebframework.shell	
Java Package	**Description**
org.carewebframework.shell	These are the core classes that comprise the shell.
org.carewebframework.shell.designer	Supports interactive user interface layout design.
org.carewebframework.shell.help	Provides integration with the help viewer.
org.carewebframework.shell.layout	Defines supported user interface elements and provides serialization support for layouts.

org.carewebframework.shell.plugins	**Provides support for plugin extensions.**
org.carewebframework.shell.property	**Provides methods for accessing and serializing properties of user interface elements.**
org.carewebframework.shell.themes	**Provides support for alternative themes.**

org.carewebframework.testharness

Java Package	Description
org.carewebframework.testharness	**The test harness controller class.**

org.carewebframework.testharness.webapp

Java Package	Description
org.carewebframework.testharness.webapp	**A minimal web application for demonstrating and testing basic CWF capabilities.**

org.carewebframework.ui

Java Package	Description
org.carewebframework.ui	**Provides core interactions with the ZK Framework and web application infrastructure.**
org.carewebframework.ui.action	**Allows the creation and registration of various types of actions that may be invoked by UI elements.**
org.carewebframework.ui.command	**Provides centralized support for keyboard shortcuts.**
org.carewebframework.ui.event	**Coordinates the delivery of generic events within the ZK environment.**
org.carewebframework.ui.icons	**A centralized registry for shared icon libraries.**
org.carewebframework.ui.spring	**Integrates Spring Framework with the ZK environment.**
org.carewebframework.ui.util	**A collection of general-purpose utilities.**
org.carewebframework.ui.xml	**A viewer for XML content.**
org.carewebframework.ui.zk	**A collection of utility methods and extensions of the ZK Framework.**
org.carewebframework.ui.thread	**Provides thread process support.**

org.carewebframework.ui.currentdatetime

Java Package	Description
org.carewebframework.ui.currentdatetime	**UI plugin to display the current date and time.**

org.carewebframework.ui.eventtesting

Java Package	Description
org.carewebframework.ui.eventtesting	**UI plugin for testing event delivery.**

org.carewebframework.ui.filetailer	
Java Package	**Description**
org.carewebframework.ui.filetailer	UI plugin for real-time monitoring of log files.
org.carewebframework.ui.icons	
Java Package	**Description**
org.carewebframework.ui.icons	The Silk icon library.
org.carewebframework.ui.infopanel	
Java Package	**Description**
org.carewebframework.ui.infopanel	UI plugin for displaying informational content.
org.carewebframework.ui.jmstesting	
Java Package	**Description**
org.carewebframework.ui.jmstesting	UI plugin for testing JMS-based message delivery.
org.carewebframework.ui.popupsupport	
Java Package	**Description**
org.carewebframework.ui.popupsupport	Supports the display of popup messages.
org.carewebframework.ui.sessiontracker	
Java Package	**Description**
org.carewebframework.ui.sessiontracker	UI plugin for monitoring active sessions and desktops.
org.carewebframework.ui.settings	
Java Package	**Description**
org.carewebframework.ui.settings	Base support for creating UI plugins for configuring settings.
org.carewebframework.ui.sharedforms	
Java Package	**Description**
org.carewebframework.ui.sharedforms	A library of reusable form styles.
org.carewebframework.ui.statuspanel	
org.carewebframework.ui.statuspanel	A UI plugin for displaying status information.
org.carewebframework.ui.wonderbar	
Java Package	**Description**
org.carewebframework.ui.wonderbar	A ZK combo box style auto-completion component.

Spring Framework Integration and Extensions

The CWF relies heavily on the Spring Framework for managing component lifecycle and scope and for wiring of interdependencies. As was noted in the preceding Spring Framework tutorial, CWF extends the capabilities of Spring in several ways, including namespace extensions for more concise declaration of plugin definitions, help and theme modules, each described in their respective sections. Additional extensions provide more granular control over overriding bean declarations and the processing order of bean configuration resources, and introduce a new bean scope, **`desktop`**. These important enhancements merit further discussion here.

Spring Configuration Files

The CWF will automatically find your Spring configuration files and process them. For this to work properly, it does impose some constraints on how you name these files and where they are located. First, all Spring configuration files must be located in the **`META-INF`** folder of the jar file. This is the same location where the jar's manifest file is located. Second, configuration files must end in "**`-spring.xml`**" in order to be discovered by Spring. How you prefix these file names is up to you, but we recommend that you use a mnemonic identifier that is not likely to be used elsewhere. Naming collisions for configuration files are only problematic in certain development environments, but should generally be avoided when possible. For example, the configuration file for a flow sheet plugin might be named **`flowsheet-spring.xml`** and would be located in the **`META-INF`** folder of that plugin's jar. The CWF would process discover and process the file during web application startup (processing declarations within the root profile) and each time a user establishes a new desktop session (processing declarations within the desktop profile).

Bean Definition Overrides

Your may have noticed the use of the **`cwf`** namespace qualifier in the example configuration file above. The CWF has implemented several namespace extensions to the Spring Framework. Some of these are to simplify the syntax for declaring certain types of beans (specifically, plugin definitions, help modules, and themes). These will be discussed in their respective sections later. The **`cwf`** namespace qualifier is different in that it actually extends the syntax of a bean declaration itself. Its purpose is to provide more granular control over the overriding of bean declarations. To understand it, a brief discussion of bean overriding in Spring is in order.

By default, Spring allows multiple bean declarations using the same bean id. Later declarations trump earlier ones. This capability allows application developers to supply differing implementations of key components. This also implies that the order of processing of configuration files is important. Spring does allow a container to be configured to reject any declarations with duplicate bean ids. In this case, encountering an id collision results in an exception being thrown. This situation does not provide fine-grained control over the overriding of bean definitions. On the one hand, allowing overrides introduces the risk of concealing unintentional id collisions. On the other hand, disabling all overrides severely

restricts the ability to flexibly configure the application. To complicate this further, bean overrides are sensitive to the order in which configuration files are processed and this order is indeterminate in the default implementation.

To provide more fine-grained control of bean overrides, the CWF provides a modified implementation of Spring's IOC container coupled with an enhanced bean declaration syntax (the **`cwf`** namespace extensions). First, the modified container guarantees that configuration files will be processed in a predictable order across all operating systems by sorting them lexicographically by file name prior to processing. This means that one can control the order of processing by adjusting the file name accordingly. You will see some configuration files with a prefix like "@001002@", for example. This ensures that these configuration files are processed before those come later lexicographically. Second, the **`cwf`** namespace enables one to be explicit about the intent of a duplicate bean id. This namespace currently supports two attributes: **`cwf:final`** is a Boolean attribute that indicates whether the associated bean declaration can be overridden. When set to true, any attempt to override the bean will throw an exception. The second, and by far more commonly used, attribute is **`cwf:override`**. This attribute has four possible values: **`always`**, **`never`**, **`ignore`**, **`default`**. The value **`always`** indicates that this bean declaration should always override any previous one. The value **`never`** indicates that this bean declaration should never override a previous one and would throw an exception if a previous declaration had already been encountered. The value **`ignore`** behaves like **`never`** except that no exception is raised – the duplicate declaration is silently ignored. Finally, the value **`default`** indicates that the default override setting of the container should be applied. This is the same as no **`cwf:override`** attribute being specified at all. The CWF IOC container is configured to not allow overrides by default, so this is functionally identical to the **`never`** attribute value.

Desktop Scope

As noted in the Spring tutorial section, the CWF implements two levels of IOC containers: one for the root application and a child container for each ZK desktop. This creates a space for beans that are shared across the web application and one for beans localized to the individual desktop. Because of the child-parent relationship, beans within the desktop container can reference those within the root container, but never the reverse. There are times, however, when one may need to reference a desktop-scoped bean in the root container. The CWF provides a custom bean scope, called **`desktop`** that allows this kind of relationship. This allows you to declare a desktop-scoped bean in the root profile, which can then be referenced by other beans within the root profile. Under the hood, this is accomplished by substituting a special proxy object in place of the desktop-scoped bean. The proxy object transparently resolves the actual target bean instance being referenced by inferring the active desktop at the time of a method call and passing the request to the appropriate instance of the target bean. This approach incurs some overhead and is, therefore, recommended only in this special circumstance.

SpringUtil Class

The CWF provides several static convenience methods for accessing Spring services:

getAppContext [S]

org.carewebframework.api.spring.SpringUtil

Parameter	Datatype	Description
<return value>	ApplicationContext	The Spring application context. May return null if an application context cannot be found.

Returns the Spring application context (IOC container). For implementations using a container hierarchy, the child application context will be returned.

getBean [S]

org.carewebframework.api.spring.SpringUtil

Parameter	Datatype	Description
id	String	The unique id of a Spring-managed bean.
<return value>	Object	A reference to the requested bean, or null if not found.

Returns a reference to a Spring-managed bean. If the bean does not exist, null is returned.

getBean [S]

org.carewebframework.api.spring.SpringUtil

Parameter	Datatype	Description
id	String	The unique id of a Spring-managed bean.
clazz	Class<T>	The class of the requested bean.
<return value>	T	A reference to the requested bean, or null if not found.

Returns a reference to a Spring-managed bean of the specified class. If the bean does not exist or a bean with a matching type cannot be found, null is returned.

getRootAppContext [S]

org.carewebframework.api.spring.SpringUtil

Parameter	Datatype	Description
<return value>	ApplicationContext	The Spring application context.

Returns the root Spring application context (IOC container). For implementations not using a container hierarchy, this returns the same as **`getAppContext`**.

isLoaded [S]

org.carewebframework.api.spring.SpringUtil

Parameter	Datatype	Description
<return value>	boolean	True if the application context has been loaded.

May be used to determine if an application context has been loaded and initialized.

setAppContextFinder [S]

org.carewebframework.api.spring.SpringUtil

Parameter	Datatype	Description
appContextFinder	IAppContextFinder	The context finder implementation.

This sets the implementation of the context finder logic. This is normally set by the CWF during initialization. For some unit tests, it may be useful to set a custom context finder.

Component Registration and Discovery

The **`org.carewebframework.api`** package provides a number of important services. Foremost among these is component registration. Component registration has some important benefits. First, registered components can be discovered by other components via the **`findObject`** method. This is useful for scenarios where one component may want to make use of services provided by another or detect whether an optional component is present. This technique works for any registered component, whether or not it is a Spring-managed bean. From a programming perspective, it is important to not write code that assumes that another plugin is present in the environment. Component registration and discovery allows you to explicitly determine the presence of another component and behave accordingly.

Second, registered components that implement the **`IRegisterEvent`** interface will be notified of each component registration event and be given an opportunity to inspect components as they are registered. This is extremely useful for components that are event producers, enabling them to discover and automatically subscribe consumers. This mechanism is used, for example, to automatically connect context change subscribers to the corresponding managed context (a producer of context change events) and to register managed contexts with the context manager.

Component registration occurs only within the desktop scope. A component may be registered only once; subsequent attempts will be ignored. In some cases, component registration occurs automatically. For example, any Spring-managed bean declared within the desktop profile will be registered with the framework. Some base classes also take care of object registration for their subclasses (subclasses of the **`FrameworkController`** class, for example).

FrameworkUtil Class

`AppFramework` is the implementation class for component registration. The **`FrameworkUtil`** class provides several static convenience methods for accessing component registration services provided by the **`AppFramework`** class:

assertInitialized [S] org.carewebframework.api.FrameworkUtil

Asserts that the application framework has been initialized. Throws an assertion exception if this is not the case.

getAppFramework [S] org.carewebframework.api.FrameworkUtil

Parameter	Datatype	Description
<return value>	AppFramework	The active application framework.

Returns a reference to the active application framework. See the section on the **`AppFramework`** class for more information.

getAppName [S]

org.carewebframework.api.FrameworkUtil

Parameter	Datatype	Description
<return value>	String	Returns the application name (may be null).

Returns the name of the currently running application. The application name is used by some properties to store application-specific settings.

getAttribute [S]

org.carewebframework.api.FrameworkUtil

Parameter	Datatype	Description
key	String	Name of the attribute.
<return value>	Object	Value of the attribute, or null if not found.

Returns a value from the named attribute. The underlying attribute map is desktop scoped. This calls **`AppFramework.getAttribute`** internally.

getAttributes [S]

org.carewebframework.api.FrameworkUtil

Parameter	Datatype	Description
<return value>	Map <String, Object>	A reference to the framework's attribute map.

Returns a reference to the framework's attribute map. This map may be used to store and retrieve arbitrary values. The attribute map is desktop scoped. This calls **`AppFramework.getAttributes`** internally.

isInitialized [S]

org.carewebframework.api.FrameworkUtil

Parameter	Datatype	Description
<return value>	boolean	True if the framework has been initialized and is ready for use.

Returns true if the framework has been initialized and is ready for use.

setAppName [S]

org.carewebframework.api.FrameworkUtil

Parameter	Datatype	Description
value	String	The name to be associated with the currently running application.

Sets the name to be associated with the currently running application. See also **`getAppName`**.

setAttribute [S]

org.carewebframework.api.FrameworkUtil

Parameter	Datatype	Description
key	String	Name of the attribute
value	Object	Value to assign.

Assigns a value to the named attribute. If the value is null, any existing attribute value is removed. The underlying attribute map is desktop scoped. This calls **`AppFramework.setAttribute`** internally.

AppFramework Class

The **`AppFramework`** class consolidates access to component services. As with most core framework components, this is a Spring-managed bean. In implementations using a container hierarchy, it resides in the child container (i.e., it is desktop-scoped). It must be accessed via the Spring application context (aka, container) that manages it. The **`FrameworkUtil.getAppFramework`** static method described previously does this for you.

The following is a summary of the methods of interest for this class:

getAttribute

org.carewebframework.api.AppFramework

Parameter	Datatype	Description
key	**String**	**Name of the attribute.**
<return value>	**Object**	**Value of the attribute, or null if not found.**

Returns a value from the named attribute.

getAttributes

org.carewebframework.api.AppFramework

Parameter	Datatype	Description
<return value>	**Map <String, Object>**	**A reference to the framework's attribute map.**

Returns a reference to the attribute map.

setAttribute

org.carewebframework.api.AppFramework

Parameter	Datatype	Description
key	**String**	**Name of the attribute**
value	**Object**	**Value to assign.**

Assigns a value to the named attribute. If the value is null, any existing attribute value is removed.

registerObject

org.carewebframework.api.AppFramework

Parameter	Datatype	Description
object	**Object**	**The object to register.**
<return value>	**boolean**	**True if the object was newly registered. False if the object was already registered.**

Registers an object to the CWF. See the section on component registration for more detail.

unregisterObject

org.carewebframework.api.AppFramework

Parameter	Datatype	Description
object	**Object**	**The object to unregister.**
<return value>	**boolean**	**False if the object was not already registered. Otherwise, will be true.**

Unregisters an object previously registered by a call to **`registerObject`**.

findObject org.carewebframework.api.AppFramework

Parameter	Datatype	Description
class	Class<?>	Returned object must be assignment-compatible with this class. If this parameter is an interface, the returned object or one of its superclasses must implement the interface.
previousInstance	Object	Previous object returned by this method. The search will begin after this object's position in the list. If this parameter is null, the search begins at the beginning.
<return value>	Object	Registered object belonging to the specified class, or null if one is not found.

Finds a registered object belonging to the specified class. The **`previousInstance`** argument allows one to easily search for multiple object instances for the same class by successive calls to this method.

IRegisterEvent Interface

Any component that has a requirement to inspect objects as they are registered with the CWF may implement the **`IRegisterEvent`** interface. When such a component is itself registered, the CWF adds it to a list of components to be notified during component registration. Whenever the **`registerObject`** method of the **`AppFramework`** class is called, the CWF notifies each component in this list via the **`registerObject`** callback method on the interface. The notified component may then inspect the component being registered and take any appropriate action. Managed contexts, as an example, use this technique to identify and automatically add subscribers to their context change event.

Because the order of component registration is not guaranteed, there is a likelihood that a component implementing the **`IRegisterEvent`** interface may be registered after components it may wish to inspect. To accommodate this scenario, the CWF will present each currently registered component to a newly registered **`IRegisterEvent`** implementer via its **`registerObject`** method. In this way, the CWF guarantees that an **`IRegisterEvent`** implementer will have an opportunity to inspect every registered component regardless of the registration order.

See the section on component registration for additional details on this topic.

registerObject org.carewebframework.api.IRegisterEvent

Parameter	Datatype	Description
object	Object	The object being registered.

Called whenever an object is registered via a call to **`AppFramework.registerObject`**.

unregisterObject org.carewebframework.api.IRegisterEvent

Parameter	Datatype	Description
object	**Object**	**The object being unregistered.**

Called whenever an object is unregistered via a call to **`AppFramework.unregisterObject`**.

Context Management

Another important core service provided by the CWF is management of shared context. A managed context wrapper may be applied to virtually any object whose state changes need to be monitored and controlled collaboratively. Typically context-wrapped objects are domain objects such as patient, user, location or encounter, but virtually any object whose state is to be tracked can be wrapped.

Context management design in the CWF is heavily borrowed from the HL7 Clinical Context Management (CCOW) specification with some important and necessary deviations. The CCOW specification was created to allow multiple independent applications running on a common desktop to share context in a standardized fashion. The specification dictates how shared contexts are accessed, serialized, modified, and synchronized.

One of the key features borrowed from CCOW is that of context synchronization. This design uses a two-phase transactional model. When application code makes a request to change a managed context, the initial phase involves a polling of each subscriber to determine its willingness to allow the change. Should any subscriber vote to block the change, the request is denied (i.e., the transaction is canceled). This is an important deviation from the CCOW specification where the requestor can still force a context change over the objections of the subscribers. Here, only the CWF itself can force a context change (this will only occur during forced logout or if a CCOW participant is forcing the context change). Another important deviation from CCOW is that a subscriber may (under certain circumstances) interact with the user prior to issuing its vote and may modify its vote based on such an interaction. The CWF provides a clear indication to the subscriber when such user interaction is permissible.

Following the outcome of the initial polling phase of a context change request, the second notification phase notifies subscribers of the outcome of the request, either committed (the subscriber may then read the updated context and act accordingly) or canceled. The result is a very seamless experience for the user.

IContextEvent Interface

Every managed context must declare (NOT implement) an extension to the **`IContextEvent`** interface, thereby creating a unique subinterface for signaling changes in the enclosed context to subscribers. Context subscribers will implement the subinterface in order to be notified of changes to the context state. See the section on writing context-aware plugins for more details.

The **`IContextEvent`** interface has the following signature:

pending

org.carewebframework.api.context.IContextEvent

Parameter	Datatype	Description
silent	**boolean**	**If true, no user interaction is permitted during the context change transaction. This will be the case during a forced logout or during a context change request originating from an external CCOW-compliant context manager. If false, the application may solicit input from the user to determine the proper course of action.**
<return value>	**String**	**If null or an empty string, signifies an affirmative vote to the requested context change. Otherwise, signifies a negative vote in which case the return value should contain a brief description of the reason. In a CCOW-managed environment, this value will be returned to the CCOW-compliant context manager, which will typically display the message to the user.**

In the first phase of a context change transaction, the context manager will invoke this method on each subscriber to indicate a pending context change is underway, giving the subscriber an opportunity to influence the outcome of the request. If any subscriber returns a negative vote, further polling stops and the context change request is canceled.

canceled

org.carewebframework.api.context.IContextEvent

Signals to the subscriber that the pending context change has been canceled. Only subscribers previously notified of a pending change (via the **`pending`** method) will receive this callback. During this time, both a current and a pending context state exist and the subscriber may inspect either via calls to the managed context wrapper.

committed

org.carewebframework.api.context.IContextEvent

Signals to the subscriber that the pending context change has now been committed. The subscriber will typically read the new context state and update its state to reflect the change. During this time, only a current context state exists and may be accessed via the managed context wrapper. Attempts to return a pending context state should always return null.

There are two programming tasks that relate to context management: writing a managed context wrapper (i.e., a context publisher) and writing context-aware code (a context subscriber). Writing a managed context wrapper is slightly more involved, but also a much less common need since most common domain objects will already have such wrappers. A much more routine task is writing context-aware code. Fortunately, this task is very straightforward. Both tasks are described in detail later in this document.

Event Management

The CWF provides simple, yet powerful support for application events based on a subscribe/publish model. Specific features of this model include

- On-the-fly creation of new event types
- Dynamic subscription
- Hierarchical event model
- Local and global event delivery
- Targeted global event delivery

Under the CWF event model, unique identifiers distinguish specific event types, rather than distinct event classes, as is the case in some models. Event operations, whether firing an event or subscribing to one, occur via these unique identifiers. The naming of event identifiers is up to the developer, but some form of namespacing is recommended to avoid naming collisions. There is no current requirement to formally register event identifiers.

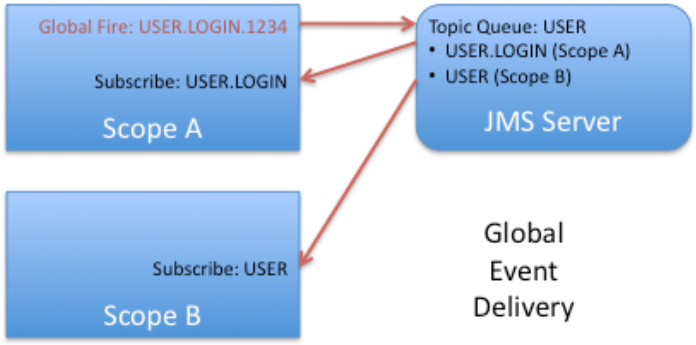
Event identifiers are hierarchical. This feature allows a component to subscribe to an event at any level of specificity. Hierarchical event identifiers are specified by separating the components of the identifier with periods. For example, the event identifier **`USER.LOGIN.1234`** specifies a hierarchical construct with **`USER`** at the top of the hierarchy and **`1234`** at the bottom. Subscribers to such an event have three levels of specificity from which to choose, **`USER`**, **`USER.LOGIN`**, or **`USER.LOGIN.1234`**. A subscriber to **`USER`** would be notified of all events in the **`USER`** hierarchy, including these: **`USER.LOGIN.4321`**, **`USER.LOGIN.1234`** and **`USER.PASSWORD.CHANGE`**. A subscriber to **`USER.LOGIN`** would receive only the first two. In this way, event producers can publish fine-grained events while a subscriber can subscribe at a less granular level if desired.

When firing an event, two parameters are typically specified. The first, the event identifier, is required. The second is the object associated with the event. The type of object is entirely up to the event producer and may be null. An event subscriber receives notification of an event on its callback interface (**`IGenericEvent`**) and is provided both the event identifier and associated object.

Events may be fired locally or globally. The delivery of locally fired events is restricted to subscribers within the same application instance. Local events are often used for inter-component communication. Globally fired events, on the other hand, are delivered to all subscribers of the event, including other application instances and potentially subscribers external to the JVM.

An implementation of global event delivery using Apache's ActiveMQ message broker, a Java Messaging Service (JMS)-compliant implementation, is provided, but other implementations may potentially be used. The advantage of the JMS-based solution is that it enables the CWF to connect to an external JMS server as well to achieve bidirectional event propagation that can extend beyond the boundaries of the web server application. Event subscription requests are automatically

forwarded to the ActiveMQ message broker. Each globally fired event is passed to the broker for delivery to its subscribers. A dedicated JMS queue (topic) is automatically created for each top-level event identifier.



Besides the scope of delivery, global events differ from local events in a couple of important ways. First, one can target specific subscribers when firing a global event. By default, a global event is delivered to all subscribers. This capability allows the event producer to exercise some level of control over the scope of delivery. Second, any object associated with a global event must be fully serializable. This requirement is not imposed on objects associated with local events.

EventUtil Class

The **`EventUtil`** class provides static convenience methods for event-related operations.

getEventManager [S]

org.carewebframework.api.event.EventUtil

Parameter	Datatype	Description
<return value>	IEventManager	The event manager.

Returns the event manager for this application context.

status [S]

org.carewebframework.api.event.EventUtil

Parameter	Datatype	Description
statusText (optional)	String	Text to be sent in the status message. If not specified, null is assumed. A null value serves to signal a subscriber that the status is being reset.

Fires a local event with an id of **`STATUS`** and an event argument that is the status text. Subscribers can listen for this event to display status updates, for example.

IEventManager Interface

This is the interface that defines the services provided to event producers and consumers by the event management infrastructure.

fireLocalEvent

org.carewebframework.api.event.IEventManager

Parameter	Datatype	Description
eventName	**String**	**The name of the event to be fired.**
eventData	**Object**	**The object to be associated with the event.**

Fires an event to all local subscribers.

fireRemoteEvent

org.carewebframework.api.event.IEventManager

Parameter	Datatype	Description
eventName	**String**	**The name of the event to be fired.**
eventData	**Object**	**The object to be associated with the event. The object must be serializable.**
recipients **(optional)**	**String**	**A comma-delimited list of client identifiers as assigned by the underlying messaging service implementation. Defaults to all subscribers.**

Fires a global event to a specific list of subscribers or to all subscribers. If a recipient has no subscribers for the named event, the event delivery request is ignored. If the recipient list is null or empty, the event is delivered to all subscribers.

hasSubscribers

org.carewebframework.api.event.IEventManager

Parameter	Datatype	Description
eventName	**String**	**The name of the event being queried.**
exact **(optional)**	**boolean**	**If true, exclude subscriptions to parent events in the query. Defaults to false.**
<return value>	**boolean**	**True if the specified event has any local subscribers.**

Determines if the named event (or any of its parent events if a hierarchical name and **`exact`** is false) has any local subscribers.

subscribe

org.carewebframework.api.event.IEventManager

Parameter	Datatype	Description
eventName	**String**	**The name of the event for which a subscription is being requested.**
subscriber	**IGenericEvent**	**The event handler.**

Establishes the specified subscriber as a subscriber to the named event.

unsubscribe

org.carewebframework.api.event.IEventManager

Parameter	Datatype	Description
eventName	**String**	**The name of the event for which a subscription is being retracted.**
subscriber	**IGenericEvent**	**The event handler.**

Removes a subscription for the specified subscriber. If no such subscription exists, the request is ignored.

IGenericEvent Interface

This is the callback interface for receiving generic events. Objects that subscribe to a generic event must pass an event handler instance that implements this interface.

eventCallback

org.carewebframework.api.event.IGenericEvent

Parameter	Datatype	Description
eventName	**String**	**The name associated with the event.**
eventData	**T**	**The object associated with the event. The object type is defined by the event producer.**

Passes event-related information to a subscriber.

Thread Management

The CWF provides support for managing background threads. It can make use of thread pooling implementations (default implementations are provided but may be overridden with custom ones). In addition, ZK imposes certain constraints on background threads that need to interact with UI components. CWF provides a **`ZKThread`** class that respects these constraints.

ThreadUtil Class

The **`ThreadUtil`** class provides several static convenience methods for thread management.

getTaskExecutor [S] org.carewebframework.api.thread.ThreadUtil

Parameter	Datatype	Description
<return value>	ExecutorService	A reference to the task executor used by the CWF, or null if none has been defined.

Returns a reference to the task executor, or null if none has been defined. The CWF is preconfigured to use Spring's **`ThreadPoolExecutorFactoryBean`** implementation.

getTaskScheduler [S] org.carewebframework.api.thread.ThreadUtil

Parameter	Datatype	Description
<return value>	ScheduledExecutorService	A reference to the scheduled executor service, or null if none has been defined.

Returns a reference to the scheduled executor service, or null if none has been defined. The CWF is preconfigured to use Spring's **`ScheduledExecutorFactoryBean`** implementation.

startThread [S] org.carewebframework.api.thread.ThreadUtil

Parameter	Datatype	Description
thread	Thread	The thread instance to be started.

Starts the given thread. If an executor service is available, it will be used to start the thread. Otherwise, the thread's start method is called directly.

Background Threads and ZK

Often times to avoid blocking user interaction, the UI needs to run a lengthy operation in the background and be notified when the operation is complete. In ZK, UI operations may only occur within a dedicated event thread. Many ZK functions rely on thread local variables to determine certain aspects of the current environment, such as the ZK desktop being serviced by the event thread. The CWF also uses this approach to associate a Spring child context with the event thread. When a ZK event thread delegates work to a background thread, this thread-based contextual information is no longer accessible. The **`ZKThread`** class was designed to address this limitation by automatically copying necessary thread-based contextual

information to the background thread and providing a means for the completed background thread to notify the foreground thread that created it.

ZKRunnable Interface

To use the **`ZKThread`** class, you must first encapsulate your background execution logic in a class that implements the **`ZKRunnable`** interface and pass an instance of this to the constructor of the **`ZKThread`** class. The **`ZKRunnable`** interface declares the following methods:

run — org.carewebframework.ui.thread.ZKThread.ZKRunnable

Parameter	Datatype	Description
thread	**ZKThread**	**The `ZKThread` instance that is executing.**

This is the execution entry point for the background thread and includes any of the program logic to be performed by the background thread. It receives the instance of the **`ZKThread`** class in which it is running. It can use this instance to convey the results of its operation back to the thread that created it (via the **`setAttribute`** method).

abort — org.carewebframework.ui.thread.ZKThread.ZKRunnable

The background thread will call this method if it has been asked to abort. The implementing class can use this to perform any necessary cleanup prior to the thread's termination.

ZKThread Class

The **`ZKThread`** class has the following methods:

ZKThread [C] — org.carewebframework.ui.thread.ZKThread

Parameter	Datatype	Description
target	**ZKRunnable**	**The target that defines the programming logic to be executed in the background.**
requester	**Component**	**The ZK component that will receive the notification that the background thread has finished.**
eventName	**String**	**The name of the event that will be sent to the requester upon background thread termination.**

Upon invoking its **`start`** method, the **`ZKThread`** instance will create a background thread and execute the **`run`** method on the **`ZKRunnable`** interface of the specified target. Upon termination of the background thread, either by normal completion or a request to abort, the requester will receive notification via the specified event name with the **`ZKThread`** instance as the event data. Note that during background thread execution there is no active desktop, so any operations requiring an active desktop will throw an exception. However, the background thread is aware of the desktop's Spring application context, so calls that require this will behave as expected.

abort

org.carewebframework.ui.thread.ZKThread

Requests that the background thread be aborted and invokes the **`abort`** method on the **`ZKRunnable`** target.

getAttribute

org.carewebframework.ui.thread.ZKThread

Parameter	Datatype	Description
name	**String**	**The name of the attribute whose value is to be retrieved.**
<return value>	**Object**	**The value of the named attribute, or null if not found.**

Returns the value of the attribute that is associated with the background thread. This can be used to convey information to the foreground thread. This method is thread-safe.

getElapsed

org.carewebframework.ui.thread.ZKThread

Parameter	Datatype	Description
<return value>	**long**	**The execution time of the background thread in milliseconds.**

Returns the execution time in milliseconds of the background thread.

getException

org.carewebframework.ui.thread.ZKThread

Parameter	Datatype	Description
<return value>	**Throwable**	**The exception thrown by the background thread, or null if there was none.**

Returns the exception thrown by the background thread.

isAborted

org.carewebframework.ui.thread.ZKThread

Parameter	Datatype	Description
<return value>	**boolean**	**True if the background thread has been asked to abort.**

Returns true if the background thread has been asked to abort.

rethrow

org.carewebframework.ui.thread.ZKThread

Rethrows the background thread's saved exception in the current thread. No action is taken if the background thread has no saved exception.

setAttribute

org.carewebframework.ui.thread.ZKThread

Parameter	Datatype	Description
name	**String**	**The name of the attribute whose value is to be set.**
value	**Object**	**The value of the named attribute.**

Sets the value of the attribute that is associated with the background thread. This operation is thread-safe.

start

org.carewebframework.ui.thread.ZKThread

Starts the background thread.

Drag and Drop Support

The ZK Framework provides general drag-and-drop support, but this alone is not sufficient to implement drag-and-drop in a generic way. For example, ZK allows for a visual element to be dragged to a recipient, but does not define what that recipient should do with the element when it is dropped. To avoid varied implementations for this common need, CWF provides some simple interfaces to deal with this in a more generic way: the **`IDropHandler`** interface to be implemented by targets of a drop event and the **`IDropRenderer`** interface to allow a dropped item to describe how it should be handled. The **`DropUtil`** utility class is provided to assist with these operations.

The ZK Framework supports drag-and-drop via the concept of drop ids. A target component will accept only dropped components that share a common drop id. Drop ids are set for target components using the **`droppable`** property while those for dragged components are set using the **`draggable`** property. See the ZK documentation for more detail.

The **`InfoPanel`** plugin is an example of how generic drag-and-drop support can be leveraged within the CWF.

DropUtil Class

The **`DropUtil`** class contains static methods that facilitate associating a drop renderer with the component that will be dropped. In this way, the component that is being dropped can convey to a drop handler how it should be rendered.

getDropRenderer org.carewebframework.ui.zk.DropUtil

Parameter	Datatype	Description
component	Component	The component whose associated drop renderer is sought.
<return value>	IDropRenderer	The drop renderer associated with the specified component, or null if there is no such association.

Returns the drop renderer associated with the specified component. Always use **`setDropRenderer`** to establish this association to ensure that the specifics of this association remain encapsulated. If a drop renderer is not associated with the specified component, this method will search its parent, if one exists, and so on until an association is found or no parent component exists. This permits associating a drop renderer at the level of a list box, for example, rather than for each list item.

setDropRenderer org.carewebframework.ui.zk.DropUtil

Parameter	Datatype	Description
component	Component	The component to associate with the specified drop renderer.
dropRenderer	IDropRenderer	The drop renderer to associate with the specified component. If a drop renderer is already associated with the component, this one replaces it. If this parameter is null, any existing drop renderer association is removed.

Use this method to associate or dissociate a drop renderer with a component.

IDropHandler Interface

The **`IDropHandler`** interface is to be implemented by handlers of a drop event. It allows a handler to describe what kind of drop events it supports (via drop ids) and a method for handling the supported drop events.

drop org.carewebframework.ui.zk.IDropHandler

Parameter	Datatype	Description
droppedItem	Component	The item being dropped. This item will typically have an associated drop renderer (see `DropUtil` class).

This method provides access to the handler's implementation. It might be called in response to a drop event initiated by user action or directly to simulate a drop event.

getDropId org.carewebframework.ui.zk.IDropHandler

Parameter	Datatype	Description
<return value>	String	A comma-delimited list of drop id(s) supported by this handler.

Returns a list of drop events supported by the handler in the form of drop ids. Use commas to separate multiple drop ids.

IDropRenderer Interface

The **`IDropRenderer`** interface embodies the logic for rendering a dropped item within the drop recipient. Because the drop renderer is associated with the item being dropped, the drop recipient does not need to know how to render the item and simply delegates this task to the drop renderer.

getDisplayText

org.carewebframework.ui.zk.IDropRenderer

Parameter	Datatype	Description
droppedItem	**Component**	**The item being dropped.**
<return value>	**String**	**Text to be displayed in association with the dropped item. This is typically a concise description of the item.**

Returns text that concisely describes the item being dropped. It may be null and may or may not be ignored by the drop handler.

isEnabled

org.carewebframework.ui.zk.IDropRenderer

Parameter	Datatype	Description
<return value>	**boolean**	**If false, the drop renderer is disabled and should not participate in drop events.**

A drop renderer may return a value of false to disable its participation in drop events.

renderDroppedItem

org.carewebframework.ui.zk.IDropRenderer

Parameter	Datatype	Description
droppedItem	**Component**	**The item being dropped.**
<return value>	**Component**	**The root component of the rendered view. The implementation may return null, indicating an inability to render the dropped item for any reason.**

The drop renderer should fully render the dropped item and return its root component. The caller will attach the returned root component to an appropriate parent.

Actions

Actions are scripts that may be executed or web resources that may be displayed by ZK components such as buttons or menu items upon receipt of a specified trigger event (by default, the **`onClick`** event). An action is referenced by its unique identifier and may be registered either locally or globally with the Action Registry to make it accessible to components running within the CWF.

IAction Interface

Every action must implement this interface.

getLabel — org.carewebframework.ui.action.IAction

Parameter	Datatype	Description
<return value>	String	The label associated with the action. This may be a label identifier prefixed with '@', or the label text itself. A component bound to this action may display this label (e.g., menu text or button label).

Returns the label associated with the action.

getScript — org.carewebframework.ui.action.IAction

Parameter	Datatype	Description
<return value>	String	The script associated with the action.

Returns the script associated with the action.

isDisabled — org.carewebframework.ui.action.IAction

Parameter	Datatype	Description
<return value>	boolean	Returns true if the action is disabled.

A disabled action will not execute even if the trigger event is received. A bound component may wish to change its visual appearance to indicate that the associated action has been disabled.

toString — org.carewebframework.ui.action.IAction

Parameter	Datatype	Description
<return value>	String	Returns displayable text for the action.

Returns the displayable text associated with the action.

ActionType Enum

This enumeration defines the type of actions that are possible. The prefix determines the action type given the text of the action script.

Value	Prefix	Description
URL	http: https:	A url reference to web content that will be displayed upon invocation of the action.
ZUL	~./	A reference to a zul page (via the ~./ path) containing zscript to be executed.
ZSCRIPT	zscript:	Zscript to be executed.
JSCRIPT	jscript: javascript:	JavaScript to be executed on the client.
UNKNOWN		The action contains an unrecognized prefix.

getType [S] org.carewebframework.ui.action.ActionType

Parameter	Datatype	Description
script	String	The script to be invoked. Must begin with one of the prefixes listed above.
<return value>	ActionType	The action type as determined by the script prefix.

Returns the action type given the script text.

ActionListener Class

The **`ActionListener`** class is a ZK event listener that allows a ZK component to invoke an action when triggered by a specified event (typically an **`onClick`** event).

addAction [S] org.carewebframework.ui.action.ActionListener

Parameter	Datatype	Description
component	Component	The component to be associated with the action.
action	IAction	The action to be invoked upon receipt of the trigger event.
eventName (optional)	String	The name of the trigger event (defaults to `onClick`).

Binds an action to a component, causing the action to be invoked when the specified trigger event is received.

addAction [S] org.carewebframework.ui.action.ActionListener

Parameter	Datatype	Description
component	Component	The component to be associated with the action.
action	String	The action to be invoked upon receipt of the trigger event. This may be action script or the name of a registered action.
eventName (optional)	String	The name of the trigger event (defaults to `onClick`).

Binds an action to a component, causing the action to be invoked when the specified trigger event is received. The **`action`** parameter may be the script text itself or the name of a registered action.

disableAction [S] org.carewebframework.ui.action.ActionListener

Parameter	Datatype	Description
component	**Component**	**The component associated with the action.**
eventName (optional)	**String**	**The name of the trigger event (defaults to `onClick`).**
disable	**boolean**	**If true, disables the action, preventing it from executing even if the trigger event is received.**

Disables or enables the triggering of an action.

removeAction [S] org.carewebframework.ui.action.ActionListener

Parameter	Datatype	Description
component	**Component**	**The component associated with the action.**
eventName (optional)	**String**	**The name of the trigger event (defaults to `onClick`).**

Unbinds an action from a component.

ActionRegistry Class

The Action Registry permits the registration of an action in either a global (web application) or local (current desktop) scope. The advantage of action registration is twofold. First, it facilitates reuse of commonly used scripts. Second, registered actions may be displayed by the Layout Designer for properties that accept actions.

addGlobalAction [S] org.carewebframework.ui.action.ActionRegistry

Parameter	Datatype	Description
action	**IAction**	**An action to be registered.**

Registers an action in the global scope. The action's label is used as the unique identifier.

addGlobalAction [S] org.carewebframework.ui.action.ActionRegistry

Parameter	Datatype	Description
label	**String**	**The action's label.**
script	**String**	**The action script.**

Registers an action in the global scope. The action's label is used as the unique identifier.

addLocalAction [S] org.carewebframework.ui.action.ActionRegistry

Parameter	Datatype	Description
action	**IAction**	**An action to be registered.**

Registers an action in the local scope. The action's label is used as the unique identifier.

addLocalAction [S] org.carewebframework.ui.action.ActionRegistry

Parameter	Datatype	Description
label	String	The action's label.
script	String	The action script.

Registers an action in the local scope. The action's label is used as the unique identifier.

removeGlobalAction [S] org.carewebframework.ui.action.ActionRegistry

Parameter	Datatype	Description
action	IAction	An action to be unregistered.

Unregisters an action from the global scope.

removeGlobalAction [S] org.carewebframework.ui.action.ActionRegistry

Parameter	Datatype	Description
label	String	The action's label.

Unregisters an action from the global scope.

removeLocalAction [S] org.carewebframework.ui.action.ActionRegistry

Parameter	Datatype	Description
action	IAction	An action to be unregistered.

Unregisters an action from the local scope.

removeLocalAction [S] org.carewebframework.ui.action.ActionRegistry

Parameter	Datatype	Description
label	String	The action's label.

Unregisters an action from the local scope.

ActionUtil Class

The **`ActionUtil`** class provides a static convenience method for creating actions.

createAction [S] org.carewebframework.ui.action.ActionUtil

Parameter	Datatype	Description
label	String	The action's label.
script	String	The action script.
<return value>	IAction	The newly created action.

Creates an action from the specified parameters.

Keyboard Shortcuts

The ZK Framework provides native support for keyboard shortcuts. It does this by allowing a component to declare which key combinations it wishes to intercept (via the **`CtrlKeys`** property). By providing a handler to process the associated key press event, the program can execute special logic when the shortcut key combination is typed. While this approach works well for monolithic applications, it does not in modular frameworks where the possibility of conflicting or inconsistent shortcut assignments is a real possibility. The CWF addresses this problem by:

- Introducing the concept of the *command* as an intermediary between a component and a keyboard shortcut. Components and keyboard shortcuts are independently bound to commands, not directly to one another.
- Externalizing and centralizing the binding of keyboard shortcuts to commands via property settings. This provides control over shortcut assignments at the web application level, not at the individual plugin level.
- Binding components to commands, not directly to keyboard shortcuts. Plugin developers do not need to be concerned with specific shortcut assignments when designing their code.
- Using ZK's native shortcut handling capability.

Applying the above, the procedure for making a component respond to a specific shortcut key combination is as follows:

- Create a command via a unique identifier that is descriptive of the function to perform (e.g., **`pageup`**). In most cases, the command is created automatically when it is first referenced by its unique identifier.
- In an external property file (**`shortcut.properties`** by default), declare any keyboard shortcut bindings to the command. A description of the format of these property declarations may be found below.
- Using one of the command binding APIs, associate your component with the desired command. These APIs are documented below.
- Write an **`onCommand`** event handler to respond to the command when it is triggered by the associated keyboard shortcut.

Using the concept of a command as an intermediary in the binding of a keyboard shortcut to the component that is to respond when the shortcut key combination is entered greatly reduces the problem of conflicting or inconsistent assignments by providing a more rational and decoupled approach to managing these assignments. Programmers can focus on implementing the functionality of the command and not be concerned about which key combination should be used to trigger the command. In addition, shortcut bindings can easily be changed with no modifications to code.

CommandUtil Class

The principal API for managing commands may be found in the **`CommandUtil`** static class. The following public methods will be of interest to the developer:

associateCommand [S]

org.carewebframework.ui.command.CommandUtil

Parameter	Datatype	Description
commandName	**String**	**The unique identifier for the command.**
component	**XulElement**	**This is the ZK component to which any associated shortcuts will be bound.**
commandTarget (optional)	**Component**	**This is the component that will receive the `onCommand` event when an associated shortcut is typed. If this parameter is absent or null, the component parameter will be the recipient of the event.**

Associates a component with a command. Internally, this binds any shortcuts associated with the command to the component and associates an **`onKeyPress`** event listener to the component. The function of this internal event listener is to intercept the shortcut key press event and send it to a target component as an **`onCommand`** event. This target component is typically the bound component itself, but may optionally be a different component. The class of the event object associated with an **`onCommand`** event is **`CommandEvent`** and is documented below.

dissociateAll [S]

org.carewebframework.ui.command.CommandUtil

Parameter	Datatype	Description
component	**XulElement**	**This is the ZK component from which any associated commands will be unbound.**

This removes any existing command associations from a component. This should be done, for example, if a bound component is about to be detached from the UI.

dissociateCommand [S]

org.carewebframework.ui.command.CommandUtil

Parameter	Datatype	Description
commandName	**String**	**The unique identifier for the command.**
component	**XulElement**	**This is the ZK component from which any associated shortcuts will be unbound.**

This removes any existing association between a command and a component.

CommandEvent Class

Event objects associated with an **`onCommand`** event are of the **`CommandEvent`** class (a subclass of the ZK Event class). The **`CommandEvent`** class declares the following methods:

getCommandName

org.carewebframework.ui.command.CommandEvent

Parameter	Datatype	Description
<return value>	**String**	**The unique identifier for the command.**

Returns the unique identifier for the command.

getReference org.carewebframework.ui.command.CommandEvent

Parameter	Datatype	Description
<return value>	Component	This is the ZK component that initiated the triggering event.

Returns the ZK component that initiated the triggering event. This is typically the component that holds the UI focus when the shortcut is typed. It may also be null.

getTriggerEvent org.carewebframework.ui.command.CommandEvent

Parameter	Datatype	Description
<return value>	Event	This is the original event that triggered the `onCommand` event.

Returns the original event that triggered the command event. This is currently always an instance of the **`KeyEvent`** class, but could potentially be of another **`Event`** subclass in future versions. It may also be null.

Shortcut Assignments

Shortcut assignments are accomplished via the **`shortcut.properties`** property file (you may locate these elsewhere if you wish by specifying a different location in the **`org.carewebframework.ui.command.shortcuts`** property in the main property file). The format of these assignments is one physical line per command, with a valid line terminator separating the physical lines. Each physical line has the format:

`<command name>=<shortcut assignments>`

where **`command name`** is the unique identifier of the command and **`shortcut assignments`** is a concatenated list of one or more shortcuts to be associated with the command using ZK's syntax for specifying shortcuts. For example:

```
foo=#f2
bar=^a@b
```

Note that more than one shortcut assignment can be specified on one line, but a given command should be specified on one and only one line (otherwise, subsequent declarations would overwrite previous assignments). You may also assign a given shortcut to different commands. Thus, the above declaration contains three assignments: one binding the command "foo" to the "F2" function key shortcut, the second binding the command "bar" to the control-A key and alt-B key shortcuts. Note that the CWF always reserves the "F1" function key shortcut, associating it with the "help" command. This assignment is implicit and need not be declared here.

Plugin Command Bindings

A plugin may declare a command association in its plugin definition declaration (see the discussion on developing plugins). This is useful if a plugin wishes to respond to

a shortcut key press but may not have the input focus. If a registered shortcut key combination is pressed when no UI element has the focus, the CWF will pass the **`onCommand`** event to any active plugins that have declared a command binding for the associated command(s). In such a case, the **`onCommand`** event is delivered to all top-level components in the plugin's main zul page. If more than one plugin is active and bound to the command, the **`onCommand`** event will be sent to each one. This behavior can be altered by calling the **`stopPropagation`** method on the **`CommandEvent`** object, which will prevent further propagation of the event.

Common Utility Library

The **`org.carewebframework.common`** package is a collection of utility classes for performing common functions.

ColorUtil Class

This is a utility class of static methods for performing color-related operations. For methods that accept or return color names, the standard set of HTML colors is used.

getNameFromRGB [S]

org.carewebframework.common.ColorUtil

Parameter	Datatype	Description
value	String	The RGB value of the color in HTML notation (e.g., #FFFAFA).
<return value>	String	The name equivalent, or null if none.

Returns the named color from an RGB string.

getRGBFromName [S]

org.carewebframework.common.ColorUtil

Parameter	Datatype	Description
name	String	The name of the color (case-insensitive).
<return value>	String	The RGB equivalent of the named color in HTML notation (e.g., #FFFAFA).

Returns the RGB equivalent of the named color.

toColor [S]

org.carewebframework.common.ColorUtil

Parameter	Datatype	Description
value	String	A color value, either in RGB format or a color name.
dflt (optional)	Color	The default Color value if the input value could not be recognized.
<return value>	Color	A Color object corresponding to the input value, or the default value (or null if no default value) if the input value was not recognized.

Converts an input value to a Color object.

DateRange Class

This class represents a date range with a start and end date.

DateRange [C]

org.carewebframework.common.DateRange

Parameter	Datatype	Description
startDate	Date	The starting date for the range.
endDate	Date	The ending date for the range.

Creates a date range instance given a starting and ending date.

DateRange [c] org.carewebframework.common.DateRange

Parameter	Datatype	Description
dateRange	DateRange	This is a copy constructor.

Creates a date range instance that is a copy of another.

DateRange [c] org.carewebframework.common.DateRange

Parameter	Datatype	Description
data	String	Date range settings in pipe-delimited format: label|start date|end date|default flag Only the label value is required.

Creates a date range instance from a string value.

DateRange [c] org.carewebframework.common.DateRange

Parameter	Datatype	Description
label	String	A text label that may be displayed in a choice list. If null, a default label is created based on the starting and ending dates.
startDate	Date	The starting date for the range.
endDate	Date	The ending date for the range.

Creates a date range instance given a label and date boundaries.

getEndDate org.carewebframework.common.DateRange

Parameter	Datatype	Description
<return value>	Date	The ending date. May be null.

Returns the ending date of the range.

getLabel org.carewebframework.common.DateRange

Parameter	Datatype	Description
<return value>	String	The label value.

Returns the value of the text label.

getRawEndDate org.carewebframework.common.DateRange

Parameter	Datatype	Description
<return value>	String	The raw string form of the ending date.

Returns the raw string form of the ending date. This may be of any recognized format, including a relative date (e.g., "`T + 1`").

getRawStartDate org.carewebframework.common.DateRange

Parameter	Datatype	Description
<return value>	String	The raw string form of the starting date.

Returns the raw string form of the starting date. This may be of any recognized format, including a relative date (e.g., "`T + 1`").

getStartDate

org.carewebframework.common.DateRange

Parameter	Datatype	Description
<return value>	Date	The starting date. May be null.

Returns the starting date of the range.

inRange

org.carewebframework.common.DateRange

Parameter	Datatype	Description
refDate	Date	A reference date for comparison.
inclusiveStart (optional)	boolean	If true, the comparison is inclusive of the starting date. Defaults to true.
inclusiveEnd (optional)	boolean	If true, the comparison is inclusive of the ending date. Defaults to false.
<return value>	boolean	True if the reference date falls within this range.

Determines if a reference date falls within the range with optional control of inclusivity.

isDefault

org.carewebframework.common.DateRange

Parameter	Datatype	Description
<return value>	boolean	True if this range is to be used as the default.

Returns true if this range is to be considered the default selection.

DateUtil Class

This is a set of static utility methods for manipulating date values. It is important to note that Java cannot distinguish a date with no time component from a date where the time is exactly midnight. For those methods that must make such a distinction, we assume that a date where all the time components (hour, minute, second and millisecond) are exactly zero represents a date with no time component.

addDays S

org.carewebframework.common.DateUtil

Parameter	Datatype	Description
date	Date	Initial date value. May be null.
daysOffset	int	# of days to add to initial value. May be negative.
stripTime	boolean	If true, strip the time component from the result.
<return value>	Date	The initial date value with the added offset. If the initial value was null, null is returned.

Adds (or subtracts) the specified number of days to an input date.

changeTimeZone [S] org.carewebframework.common.DateUtil

Parameter	Datatype	Description
date	Date	Initial date value.
oldTimeZone	TimeZone	Original timezone.
newTimeZone	TimeZone	New timezone.
<return value>	Date	The date with its time adjusted from the original time zone to a new one.

Changes the time component of an input date from one time zone to another.

cloneDate [S] org.carewebframework.common.DateUtil

Parameter	Datatype	Description
date	Date	Date to clone. May be null.
<return value>	Date	A new date instance with the same date value as the original. If the original value was null, null is returned.

Clones a date object, allowing a null input.

compare [S] org.carewebframework.common.DateUtil

Parameter	Datatype	Description
date1	Date	First date to compare. May be null.
date2	Date	Second date to compare. May be null.
<return value>	Date	Result of the comparison.

Compares two dates, allowing for null inputs.

endOfDay [S] org.carewebframework.common.DateUtil

Parameter	Datatype	Description
date	Date	Initial date value.
<return value>	Date	The original date value with the time component set to the end of the day (one millisecond before midnight).

Returns an input date with the time component set to the end of the day. This is useful for date comparisons requiring an inclusive end point.

formatAge [S] org.carewebframework.common.DateUtil

Parameter	Datatype	Description
dob	Date	Date of birth.
pluralize (optional)	boolean	If true, the units in the return value will be pluralized when appropriate. Defaults to true.
refDate (optional)	Date	Reference date from which to calculate the age. Defaults to today.
<return value>	String	An age in displayable form.

Returns the calculated age in displayable format expressed in days, months or years depending on the age threshold.

formatDate [S] org.carewebframework.common.DateUtil

Parameter	Datatype	Description
date	Date	Date value to format. If null, an empty string is returned.
showTimezone (optional)	boolean	If true, the time zone is included in the output. Defaults to false.
ignoreTime (optional)	boolean	If true, the time component is omitted from the output. Defaults to false.
<return value>	String	The date formatted for display using the format: dd-MMM-yyyy HH:mm zzz

Formats a date for display using a standardized format.

formatDate [S] org.carewebframework.common.DateUtil

Parameter	Datatype	Description
date	Date	Date value to format. If null, an empty string is returned.
timeSeparator	String	Delimiter to use in place of the default time separator.
<return value>	String	Format used by above formatDate method using the default values for optional parameters, with the default time separator replaced by the one specified.

Formats a date, replacing the default delimiter separating the date from the time component with the one specified.

formatDuration [S]

org.carewebframework.common.DateUtil

Parameter	Datatype	Description
duration	long	A duration value in milliseconds.
accuracy (optional)	Accuracy	The accuracy of the returned value. This is an enumeration value that can specify an accuracy of milliseconds to years. The default is milliseconds.
pluralize (optional)	boolean	If true, the units in the return value will be pluralized when appropriate. Defaults to true.
abbreviated (optional)	boolean	If true, the abbreviated form of units are used.
<return value>	String	The duration in displayable form.

Formats a duration value in displayable form.

formatElapsed [S]

org.carewebframework.common.DateUtil

Parameter	Datatype	Description
elapsed	double	An elapsed time in milliseconds.
pluralize (optional)	boolean	If true, the units in the return value will be pluralized when appropriate. Defaults to true.
abbreviated (optional)	boolean	If true, the abbreviated form of units are used.
round (optional)	boolean	If true, round the result to an integer.
<return value>	String	The duration in displayable form.

Formats an elapsed time in displayable form.

getLocalTimeZone [S]

org.carewebframework.common.DateUtil

Parameter	Datatype	Description
<return value>	TimeZone	The local time zone.

Retrieves the local time zone from the time zone accessor as specified in the **`localTimeZone`** static variable in this class. The default implementation for the time zone accessor may be replaced with a custom one by setting this static variable to an alternative implementation.

hasTime [S]

org.carewebframework.common.DateUtil

Parameter	Datatype	Description
date	Date	The date value to check.
<return value>	boolean	True if the date has a time component.

Determines whether the date has a time component.

now [S]

org.carewebframework.common.DateUtil

Parameter	Datatype	Description
<return value>	Date	A date set to the current date/time.

Returns a date object set to the current date and time.

parseDate [S]

org.carewebframework.common.DateUtil

Parameter	Datatype	Description
value	String	A string containing some representation of a date.
<return value>	Date	The date value specified by the input string, or null if the input string could not be parsed.

Parses an input string that contains a date and time representation. A number of input formats are supported, including relative dates (e.g., "`T + 1`").

stripTime [S]

org.carewebframework.common.DateUtil

Parameter	Datatype	Description
date	Date	The initial date value.
<return value>	Date	The original date value with the time component removed.

Returns the input date without the time component.

toDate [S]

org.carewebframework.common.DateUtil

Parameter	Datatype	Description
day	int	Day of the month (starting at 1).
month	int	Month of the year (starting at 1).
year	int	Year (four digits).
hr (optional)	int	Hour (24 hour clock). Defaults to 0.
min (optional)	int	Minutes. Defaults to 0.
sec (optional)	int	Seconds. Defaults to 0.
<return value>	Date	A date with the specified settings.

This is a convenience method to easily create a date, with or without a time component, from individual settings.

today [S]

org.carewebframework.common.DateUtil

Parameter	Datatype	Description
<return value>	Date	Today's date, without a time component.

Returns today's date without a time component.

toHL7 [S] org.carewebframework.common.DateUtil

Parameter	Datatype	Description
date	Date	The input date.
<return value>	String	Returns the input date formatted according to the HL7 specification.

Returns the input date in the format prescribed by the HL7 specification.

ImageUtil Class

This is a set of static utility methods for image-related operations.

toImage [S] org.carewebframework.common.ImageUtil

Parameter	Datatype	Description
text	String	The text to render.
font (optional)	Font	The font to use to render the text. Defaults to Tahoma, plain, 11pt.
backColor (optional)	Color	The background color. Defaults to white.
fontColor (optional)	Color	The font color. Defaults to black.
<return value>	BufferedImage	The rendered image as a BufferedImage object.

Renders a text string as an image. Can be used where one might want to display a text value as a background image.

JSONUtil Class

This is a set of static utility methods providing support for JSON operations. For the association of a JSON object with the supporting Java class, we use alias names rather than the class name itself. In this way, a JSON object can identify its top-level data type without tying itself to a specific implementation class.

deserialize [S] org.carewebframework.common.JSONUtil

Parameter	Datatype	Description
data	String	Object serialized in JSON format. If null, a null value is returned.
<return value>	Object	An instance of the deserialized object.

Deserializes an object from JSON format. An exception is raised if the operation fails. This method automatically calls `deserializeList` if it detects that the source represents a list.

deserializeList [S] org.carewebframework.common.JSONUtil

Parameter	Datatype	Description
data	String	Object list serialized in JSON format.
clazz	Class<T>	The type of each list element.
<return value>	List<T>	An instance of the deserialized list.

Deserializes a list of objects.

getAlias [S] org.carewebframework.common.JSONUtil

Parameter	Datatype	Description
clazz	Class	The Java class whose alias is sought.
<return value>	String	The JSON alias corresponding to the Java class, or null if one does not exist.

This retrieves the JSON alias to be used for a specific Java class.

registerAlias [S] org.carewebframework.common.JSONUtil

Parameter	Datatype	Description
alias	String	The alias name to be registered.
clazz	Class	The Java class to be associated with the alias.

Associates an alias name with a Java class.

serialize [S] org.carewebframework.common.JSONUtil

Parameter	Datatype	Description
object	Object	An object instance to be serialized.
<return value>	String	A JSON string.

Serializes an object instance as a JSON-formatted string.

setDateFormat [S] org.carewebframework.common.JSONUtil

Parameter	Datatype	Description
dateFormat	DateFormat	Sets the date formatter to be used when serializing dates to JSON.

Provides control over how dates are serialized.

unregisterAlias [S] org.carewebframework.common.JSONUtil

Parameter	Datatype	Description
name	String	The name of the alias to unregister.

Removes a registered alias.

MiscUtil Class

This is a set of static utility methods performing miscellaneous functions.

castList [S] org.carewebframework.common.MiscUtil

Parameter	Datatype	Description
list	List<T>	The list whose elements are to be recast.
clazz	Class<E>	Class to which list elements are to be recast.
<return value>	List<E>	The original list with elements recast.

This is a convenience method for recasting a list of elements from one type to another type that is a subclass of the original. In other words, class E must extend from class T. This method performs a simple recast and does not copy the original list.

containsInstance [S] org.carewebframework.common.MiscUtil

Parameter	Datatype	Description
list	List	The list to search.
object	Object	The object being sought.
<return value>	boolean	True if the list contains the object instance being sought.

Returns true if the source list contains the object instance. The search uses object identity, not equality.

fileExists [S] org.carewebframework.common.MiscUtil

Parameter	Datatype	Description
fileName	String	The path name of the file.
<return value>	boolean	True if the file exists.

This is a convenience method to determine if a file exists given its path name.

indexOfInstance [S] org.carewebframework.common.MiscUtil

Parameter	Datatype	Description
list	List	The list to search.
object	Object	The object being sought.
<return value>	int	The index of the object in the list, or -1 if not found.

Returns the index of the object instance within the list. The search uses object identity, not equality.

NaturalStringSorter Class

This is a set of static utility methods (compliments of Stephen Kelvin Friedrich) for performing comparisons of strings with embedded numbers such that numeric values collate as such and not as string values.

compareNatural [S] org.carewebframework.common.NaturalStringSorter

Parameter	Datatype	Description
collator (optional)	Collator	The collator implementation to use for non-digit comparison. If not specified, a collator for the default locale is used.
source1	String	First string to compare.
source2	String	Second string to compare.
<return value>	int	Result of the comparison.

Compares two string values using either the default or a supplied collator for non-digit comparison.

compareNaturalAscii [S] org.carewebframework.common.NaturalStringSorter

Parameter	Datatype	Description
source1	String	First string to compare.
source2	String	Second string to compare.
<return value>	int	Result of the comparison.

Compares two string values using the Unicode value for non-digit comparison.

compareNaturalIgnoreCaseAscii [S] org.carewebframework.common.NaturalStringSorter

Parameter	Datatype	Description
source1	String	First string to compare.
source2	String	Second string to compare.
<return value>	int	Result of the comparison.

Compares two string values without case sensitivity using the Unicode value for non-digit comparison.

getNaturalComparator [S] org.carewebframework.common.NaturalStringSorter

Parameter	Datatype	Description
collator (optional)	Collator	The collator implementation to use for non-digit comparison. If not specified, a collator for the default locale is used.
<return value>	Comparator <String>	A natural string comparator using the specified collator.

Returns a natural string comparator.

getNaturalComparatorAscii [S] org.carewebframework.common.NaturalStringSorter

Parameter	Datatype	Description
<return value>	Comparator <String>	A natural string comparator using Unicode values for non-digit comparison.

Returns a natural string comparator that uses Unicode values for non-digit comparison.

getNaturalComparatorIgnoreCaseAscii [S] org.carewebframework.common.NaturalStringSorter

Parameter	Datatype	Description
<return value>	Comparator <String>	A case-insensitive natural string comparator using Unicode values for non-digit comparison

Returns a case-insensitive natural string comparator that uses Unicode values for non-digit comparison.

NumUtil Class

This is a set of static utility methods for performing common numeric operations.

compare [S] org.carewebframework.common.NumUtil

Parameter	Datatype	Description
val1	int	First value for comparison
val2	int	Second value for comparison
<return value>	int	Result of comparison.

Compares two integer values, returning -1 if val1 < val2, 0 if val1 = val2, or 1 if val1 > val2.

compare [S] org.carewebframework.common.NumUtil

Parameter	Datatype	Description
val1	double	First value for comparison
val2	double	Second value for comparison
<return value>	int	Result of comparison.

Compares two double values, returning -1 if val1 < val2, 0 if val1 = val2, or 1 if val1 > val2.

enforceRange [S] org.carewebframework.common.NumUtil

Parameter	Datatype	Description
value	int	Value to be checked.
minValue	int	Minimum allowable value.
maxValue	int	Maximum allowable value.
<return value>	int	The adjusted value.

Forces an integer value to fall within a specified range. Returns the original value if it falls within the range, or the minimum or maximum value if it falls below or above, respectively.

toString [S] org.carewebframework.common.NumUtil

Parameter	Datatype	Description
value	double	Value to be converted.
<return value>	String	String equivalent with the trailing fractional zero.

Converts a double value to its string form without trailing fractional zeroes.

StopWatch Support

There are a number of available implementations of stopwatch functionality for tracking performance metrics. Rather than force a specific implementation, the CWF allows for the registration of an implementation of choice via the **`StopWatchFactory`** class. The **`org.carewebframework.ext.performance`** Maven artifact exposes the perf4j stopwatch implementation via this mechanism.

StopWatchFactory Class

The **`StopWatchFactory`** class declares the following static methods:

createFactory [S] org.carewebframework.common.StopWatchFactory

Parameter	Datatype	Description
clazz	Class <? extends IStopWatch>	The stopwatch implementation class to use.
<return value>	StopWatchFactory	The stopwatch factory for creating stopwatch instances using the specified implementation.

This initializes the stopwatch factory, specifying the stopwatch implementation to be used. This may be called only once.

create [S] org.carewebframework.common.StopWatchFactory

Parameter	Datatype	Description
<return value>	IStopWatch	A new, uninitialized stopwatch instance. Will return null if no stopwatch implementation has been provided.

Creates a new, uninitialized stopwatch instance.

create [S] org.carewebframework.common.StopWatchFactory

Parameter	Datatype	Description
tag	String	Tag to identify the interval being timed.
data	Map <String, Object>	Metadata to be associated with the interval being timed.
<return value>	IStopWatch	A new, initialized stopwatch instance. Will return null if no stopwatch implementation has been provided.

Creates a new stopwatch instance and calls its **`init`** method with the specified parameters.

IStopWatch Interface

Candidate stopwatch implementations must implement the **`IStopWatch`** interface. This is typically a simple wrapper around or extension to an existing stopwatch class. This interface has the following signature:

init org.carewebframework.common.StopWatchFactory.IStopWatch

Parameter	Datatype	Description
tag	**String**	**A tag to identify the interval being timed.**
data	**Map <String, Object>**	**Metadata to be associated with the interval being timed.**

Initializes the stopwatch instance. The underlying implementation may or may not support both parameters and may, therefore, choose to ignore them.

start org.carewebframework.common.StopWatchFactory.IStopWatch

Starts the stopwatch.

stop org.carewebframework.common.StopWatchFactory.IStopWatch

Stops the stopwatch.

StrUtil Class

This is a set of static utility methods for performing common string operations.

extractInt [S] org.carewebframework.common.StrUtil

Parameter	Datatype	Description
value	**String**	**The value to parse.**
<return value>	**int**	**The integer value extracted from the input.**

Extracts the integer prefix from a string, returning it as an integer value. Parsing stops when a non-digit character or the end of the string is encountered. If no integer prefix is found, returns 0.

extractIntPrefix [S] org.carewebframework.common.StrUtil

Parameter	Datatype	Description
value	**String**	**The value to parse.**
<return value>	**String**	**The integer prefix extracted from the input.**

Extracts the integer prefix from a string, returning it as a string. Parsing stops when a non-digit character or the end of the string is encountered. If no integer prefix is found, returns an empty string.

formatMessage [S]

org.carewebframework.common.StrUtil

Parameter	Datatype	Description
msg	String	Message to format or, if prefixed with "@", is assumed to be a label name to be resolved.
locale (optional)	Locale	The locale. Uses the default locale if not specified.
args (optional)	Object…	Optional replaceable parameter values.
<return value>	String	The formatted message.

Formats message text. If the message text begins with an "@" character, it is assumed to be a label name and is resolved via the registered message source (see **`setMessageSource`** in this section). If replaceable parameter values are specified, these are substituted into the corresponding placeholder positions within the input string.

fromList [S]

org.carewebframework.common.StrUtil

Parameter	Datatype	Description
list	Iterable <String>	The list of strings to concatenate.
delimiter (optional)	String	The delimiter to use to separate elements. Defaults to the newline character.
dflt (optional)	String	Default value to use if a list entry is null. Defaults to an empty string.
<return value>	String	Concatenated list of elements.

Builds a string of delimited elements from an input list. If a list entry is null and the default value is also null, the entry will be ignored.

getMessageSource [S]

org.carewebframework.common.StrUtil

Parameter	Datatype	Description
<return value>	MessageSource	The message source for resolving messages.

Returns the Spring message source to be used for resolving label references. This is normally set by the CWF configuration you are using (e.g., for the web configuration, this is set to use ZK's label resolver).

piece [S]

org.carewebframework.common.StrUtil

Parameter	Datatype	Description
text	String	The input string to be parsed.
delimiter	String	The delimiter that separates individual elements.
start (optional)	int	First delimited element to extract. Defaults to element in the first position.
end (optional)	int	Last delimited element to extract. Defaults to starting position.
<return value>	String	The extracted elements.

Returns the delimited portions of an input string.

setMessageSource [S]

org.carewebframework.common.StrUtil

Parameter	Datatype	Description
messageSource	**MessageSource**	**The message source for resolving messages.**

Sets the Spring message source to be used for resolving label references. This is normally set by the CWF configuration you are using (e.g., for the web configuration, this is set to use ZK's label resolver).

split [S]

org.carewebframework.common.StrUtil

Parameter	Datatype	Description
text	**String**	**The input string to be parsed.**
delimiter	**String**	**The delimiter that separates individual elements.**
count **(optional)**	**int**	**Specifies the minimum number of elements to return. If the result of the split operation results in fewer elements, the returned array is expanded to contain this minimum number. Defaults to 0.**
nonull **(optional)**	**boolean**	**If true, guarantees that the return array will contain no null values (null values are replaced by empty strings). Defaults to true.**
<return value>	**String[]**	**The extracted elements.**

Splits a string into constituent elements given a delimiter.

strAppend [S]

org.carewebframework.common.StrUtil

Parameter	Datatype	Description
text	**String**	**The text string to append to. If null, returns null.**
value	**String**	**The value to append. If null or empty, returns the original text.**
separator **(optional)**	**String**	**The value to separate appended text from original input.** **Defaults to a comma followed by a space.**
<return value>	**String**	**The original text with the appended value.**

Appends a value to a text string, separated by the specified separator string.

stripQuotes [S]

org.carewebframework.common.StrUtil

Parameter	Datatype	Description
value	**String**	**The input string.**
<return value>	**String**	**The input string less any enclosing quotes.**

Strips enclosing quotes (single or double) from an input string.

strTruncate [S] org.carewebframework.common.StrUtil

Parameter	Datatype	Description
value	String	The input string.
maxLength	int	The maximum length of the returned string.
<return value>	String	The input string truncated as necessary. In the event of truncation, three periods are appended to the truncated value.

Truncates an input string to the specified length.

toBoolean [S] org.carewebframework.common.StrUtil

Parameter	Datatype	Description
text	String	The input string.
<return value>	boolean	Converts a numeric text value to a Boolean result. Any non-zero numeric value is interpreted as true. Any non-numeric input is interpreted as false.

Converts a numeric text input to a Boolean value.

toInt [S] org.carewebframework.common.StrUtil

Parameter	Datatype	Description
text	String	The input string.
<return value>	int	Converts a numeric text value to an integer value. If the input string cannot be converted, returns 0.

Converts a numeric text input to an integer value.

toList [S] org.carewebframework.common.StrUtil

Parameter	Datatype	Description
text	String	The input string.
list (optional)	List <String>	The list to receive the parsed elements. If null or not specified, a new list is created. Otherwise, the existing list is cleared before adding elements.
delimiter (optional)	String	The string that delimits the individual elements in the input string. Defaults to the newline character.
<return value>	List <String>	Returns a list containing the parsed elements.

Converts a string containing delimited elements to a list.

toLong [S] org.carewebframework.common.StrUtil

Parameter	Datatype	Description
text	String	The input string.
<return value>	long	Converts a numeric text value to a long integer value. If the input string cannot be converted, returns 0.

Converts a numeric text input to a long integer value.

xlate [S] org.carewebframework.common.StrUtil

Parameter	Datatype	Description
text	String	The input string.
from	String	A string of characters to be replaced.
to	String	A string of replacement characters.
<return value>	String	A string resulting from the replacement of each occurrence of a character in the `from` parameter with the corresponding character in the `to` parameter.

Replaces each occurrence of a character in the **`from`** parameter with the corresponding character in the **`to`** parameter. If the character to be replaced has no corresponding character, it is removed from the input string.

WeakArrayList Class

This is an implementation of a list that maintains weak references to its elements. A weak reference does not prevent an element from being garbage-collected if it is otherwise eligible. An attempt to retrieve a garbage-collected element from the list via the **`get`** method will return a null value.

In addition to the standard methods for a list implementation, the following method is also available:

compact org.carewebframework.common.WeakArrayList

Removes any garbage-collected entries from the list.

XMLUtil Class

This is a set of static utility methods for manipulating XML data.

formatAttributes [S] org.carewebframework.common.XMLUtil

Parameter	Datatype	Description
node	Node	An XML document node.
<return value>	String	The node attributes as a formatted string.

Formats the attribute name-value pairs of a DOM node in the form:

```
<attribute name>='<attribute value>'
```

formatNodeName [S] org.carewebframework.common.XMLUtil

Parameter	Datatype	Description
node	Node	An XML document node.
format	TagFormat	The desired tag format. One of: • OPENING – Format as an opening tag complete with attribute values. • CLOSING – Format as a closing tag. • BOTH – Format as an opening tag followed by a closing tag. • EMPTY – Format as a self-closing tag complete with attribute values.
<return value>	String	The node as a string in the requested format.

Formats DOM node as a tag in one of the requested formats.

parseXMLFromList [S] org.carewebframework.common.XMLUtil

Parameter	Datatype	Description
xml	Iterable <String>	The XML source.
<return value>	Document	A fully parsed document.

Parses an XML document from an iterable list.

parseXMLFromLocation [S] org.carewebframework.common.XMLUtil

Parameter	Datatype	Description
filePath	String	The full path to a file containing valid XML.
<return value>	Document	A fully parsed document.

Parses an XML document from a text file.

parseXMLFromStream [S] org.carewebframework.common.XMLUtil

Parameter	Datatype	Description
stream	InputStream	An input stream containing valid XML.
<return value>	Document	A fully parsed document.

Parses an XML document from an input stream.

parseXMLFromString [S] org.carewebframework.common.XMLUtil

Parameter	Datatype	Description
xml	String	A string containing valid XML.
<return value>	Document	A fully parsed document.

Parses an XML document from a string containing valid XML.

toString [S] org.carewebframework.common.XMLUtil

Parameter	Datatype	Description
doc	Document	An XML document.
<return value>	String	A formatted string representation of the XML document.

Converts a DOM document to formatted XML text.

ZK Utility Functions

These are miscellaneous functions that facilitate or enhance existing ZK features.

MenuUtil Class

The **`MenuUtil`** class contains several static convenience methods for managing ZK menus.

addMenu [S] org.carewebframework.ui.zk.MenuUtil

Parameter	Datatype	Description
path	String	A path that describes the position of the menu item in the menu tree (see discussion below).
menu	Menu	The menu to be added. If null, a new menu is created.
menubar	Menubar	The menu bar to receive the menu.
insertBefore	Component	The sibling before which the menu is to be inserted. If null, or the specified component is not at the same level as the menu to be added, the new menu will be added after any existing siblings.
<return value>	Menu	The menu that was added. If the `menu` parameter was not null, this value is returned. Otherwise, a reference to the newly created menu is returned.

Adds an existing, or creates a new, menu to the specified menu bar. The path consists of a backslash-delimited string of menu nodes (identified by the associated text label) that represents the desired position of the menu in the menu tree. Any intervening menus that do not already exist are automatically created. For example, the path **`Help\Topics\Patient Selection`** would place a menu labeled **`Patient Selection`** at the third level of the menu tree under a menu labeled **`Topics`**, which in turn is under a top-level menu labeled **`Help`**.

addMenuItem [S] org.carewebframework.ui.zk.MenuUtil

Parameter	Datatype	Description
path	String	A path that describes the position of the menu item in the menu tree (see discussion above).
menuItem	Menuitem	The menu item to be added. If null, a new menu item is created.
menubar	Menubar	The menu bar to receive the menu item.
insertBefore	Component	The sibling before which the menu item is to be inserted. If null, or the specified component is not at the same level as the item to be added, the new menu item will be added after any existing siblings.
<return value>	Menu	The menu item that was added. If the `menuItem` parameter was not null, this value is returned. Otherwise, a reference to the newly created menu item is returned.

Adds an existing, or creates a new, menu item to the specified menu bar.

close [S] org.carewebframework.ui.zk.MenuUtil

Parameter	Datatype	Description
menu	Menu	The menu to be closed.

Closes a top-level menu popup.

findMenu [S] org.carewebframework.ui.zk.MenuUtil

Parameter	Datatype	Description
parent	Component	The parent menu component under which to initiate the search. This may be a `Menubar` or a `Menupopup` component.
label	String	The label of the menu being sought.
insertBefore	Component	If a new menu is created, this is the sibling before which the new menu is to be inserted. If null, or the specified component is not at the same level as the menu to be added, the new menu will be added after any existing siblings. This parameter is ignored if an existing menu was found.
<return value>	Menu	The menu whose label matched (case-insensitive) the specified label, or failing a match, the newly created menu with the specified label.

Returns the menu with the specified label, creating one if it does not exist. The search is case-insensitive and only searches the immediate children of the parent component.

findMenu [S] org.carewebframework.ui.zk.MenuUtil

Parameter	Datatype	Description
menubar	Menubar	The menu bar to search.
path	String	The \ - delimited path. Path nodes may be label values or node indexes, depending on the match mode.
create	boolean	If true, menus are created if they do not exist. If false, the method will return null if any menu within the path is not found.
clazz	Class <? extends Menu>	The class to use when creating new menus.
matchMode	MatchMode	The mode to use when matching a menu to a path node. • INDEX – Treat path nodes as node index values. • CASE_SENSITIVE – Treat path nodes as menu labels and match with case sensitivity. • CASE_INSENSITIVE – Treat path nodes as menu labels and match without case sensitivity. • AUTO – Automatically determine path node type (index vs. label).
<return value>	Menu	The menu matching the specified path. If the create parameter is false and the menu was not found, will return null.

Returns the menu corresponding to the specified path. The path may consist of menu label values, menu index values, or both, depending on the match mode setting.

getPath [S] org.carewebframework.ui.zk.MenuUtil

Parameter	Datatype	Description
comp	LabelImageElement	The menu or menu item whose path is to be returned.
<return value>	String	The path of the specified menu or menu item consisting of \ - delimited string of node labels.

Returns the path of the specified menu or menu item.

pruneMenus [S] org.carewebframework.ui.zk.MenuUtil

Parameter	Datatype	Description
parent	Component	The menu element to be examined and removed if it has no children.

Recursively removes empty menu container elements, beginning at the specified parent and moving up the menu tree. This method is typically used after removing menu elements to keep the menu structure lean.

sortMenu [S] org.carewebframework.ui.zk.MenuUtil

Parameter	Datatype	Description
parent	Component	Parent menu element whose children are to be sorted.
startIndex	Integer	Index of the first child to be sorted.
endIndex	Integer	Index of the last child to be sorted.

Alphabetically sorts a range of menu elements. The sorting is case-insensitive.

updateStyles [S] org.carewebframework.ui.zk.MenuUtil

Parameter	Datatype	Description
component	Component	A component in the menu tree.

The ZK component model makes a distinction between a menu element and a menu item. The principle difference is that a menu element may have children whereas a menu item may not. We find that a component model that makes no such distinction is simpler to manage. To achieve this, the CWF builds menu trees consisting only of menu elements – no menu items. This method will restyle any childless menu element as a menu item.

MoveEventListener Class

ZK allows a user to drag a floating window to a new position within the browser window (view port). If the user moves the window outside the bounds of the view port, it may become inaccessible. If it is a modal window, further user interaction becomes impossible. When added as a listener for a window's **`onMove`** event, the **`MoveEventListener`** class will prevent this scenario by ensuring that the window remains accessible within the browser's view port. Several CWF API's that produce floating windows (e.g., the **`PopupDialog`** class), apply this listener automatically.

This class has two static convenience methods for adding or removing a singleton listener.

add [S] org.carewebframework.ui.zk.MoveEventListener

Parameter	Datatype	Description
component	Component	The component to receive the event listener.

Adds a singleton instance of the **`MoveEventListener`** class to the specified component.

remove [S] org.carewebframework.ui.zk.MoveEventListener

Parameter	Datatype	Description
component	Component	The component from which the event listener will be removed.

Removes the singleton instance of the **`MoveEventListener`** class from the specified component.

PopupDialog Class

The **`PopupDialog`** class extends ZK's **`Window`** class and provides a convenient means for displaying a zul page in a popup window. It may also be easily subclassed to provide extended functionality.

close

org.carewebframework.ui.zk.PopupDialog

Parameter	Datatype	Description
canceled	**boolean**	**Sets the cancel status before closing the window.**

Closes the window, setting its cancel status prior to doing so.

isCanceled

org.carewebframework.ui.zk.PopupDialog

Parameter	Datatype	Description
<return value>	**boolean**	**The cancel status of the window, as set by the `close` method.**

Returns the cancel status of the window. This is set by the **`close`** method.

onResize

org.carewebframework.ui.zk.PopupDialog

Parameter	Datatype	Description
newHeight	**String**	**The new height of the window.**
newWidth	**String**	**The new width of the window.**

Called whenever the window is resized. The base implementation of this method does nothing. Subclasses may override it to provide specialized behavior.

popup S

org.carewebframework.ui.zk.PopupDialog

Parameter	Datatype	Description
zulPageDefinition	**PageDefinition**	**The page definition to be used to construct the dialog.**
args (optional)	**Map<Object,Object>**	**A list of arguments to be passed to the zul page when it is created. May be null (the default).**
closable (optional)	**boolean**	**If true (the default), the window will have a close button on the title bar.**
sizable (optional)	**boolean**	**If true (the default), the window may be resized by the user.**
show (optional)	**boolean**	**If true (the default), the window is displayed modally immediately after it is created. If false, the window is created, but not displayed.**
<return value>	**Window**	**The window that was created. If the top-level component of the zul page is a `Window`, that component is returned. Otherwise, a new `Window` is created to host the zul page and that `Window` is returned.**

Use this static method to popup any zul page given its definition.

popup [S] org.carewebframework.ui.zk.PopupDialog

Parameter	Datatype	Description
zulPage	**String**	**The URL of a zul page to be displayed in a popup window.**
closable (optional)	**boolean**	**If true (the default), the window will have a close button on the title bar.**
sizable (optional)	**boolean**	**If true (the default), the window may be resized by the user.**
show (optional)	**boolean**	**If true (the default), the window is displayed modally immediately after it is created. If false, the window is created, but not displayed.**
<return value>	**Window**	**The window that was created. If the top-level component of the zul page is a `Window`, that component is returned. Otherwise, a new `Window` is created to host the zul page and that `Window` is returned.**

Use this static method to popup any zul page given its URL.

show org.carewebframework.ui.zk.PopupDialog

Displays the window modally.

PopupManager Class

The Popup Manager maintains a list of active popup windows and ensures that each popup window does not fully obscure the preceding one by staggering the initial position of successive popup windows. Dialogs belonging to the **`PopupDialog`** class, for example, automatically register themselves with the Popup Manager. The following methods will be of interest to the software developer:

getInstance [S] org.carewebframework.ui.zk.PopupManager

Parameter	Datatype	Description
<return value>	**PopupManager**	**The instance of the Popup Manager for the current desktop.**

Use this static method to access the Popup Manager for the current desktop.

registerPopup org.carewebframework.ui.zk.PopupManager

Parameter	Datatype	Description
window	**Window**	**The window to be registered.**

This registers a window with the Popup Manager. This should be done before the window is displayed. This method also applies a **`MoveEventListener`** to the window to prevent the window from being moved outside the browser view port and removes the default browser context menu from the window (it will not affect a custom context menu, however).

unregisterPopup org.carewebframework.ui.zk.PopupManager

Parameter	Datatype	Description
window	**Window**	**The window to be unregistered.**

This method is called automatically when a registered popup window is closed.

PromptDialog Class

This class provides a convenient way to prompt the user for various kinds of simple input. There are a number of static methods.

confirm [S] org.carewebframework.ui.zk.PromptDialog

Parameter	Datatype	Description
message	**String**	**The text of the user prompt.**
title (optional)	**String**	**An optional title for the dialog. If none is specified, it defaults to "Confirm".**
responseId (optional)	**String**	**An optional response id if the user response is to be cached. If null or not specified, the response will not be cached. If specified, the response is cached and the user is not prompted again.**
<return value>	**boolean**	**True if the user clicked the OK button. False otherwise.**

This static method provides a convenient means to prompt the user for confirmation of an action. If a unique response id is supplied, the user's initial response to the dialog is recorded and subsequent calls to this method using the same response id will not prompt the user for input but rather return the value of the stored response.

input [S] org.carewebframework.ui.zk.PromptDialog

Parameter	Datatype	Description
message	**String**	**The text of the user prompt.**
title	**String**	**The title for the dialog.**
oldValue (optional)	**String**	**The prior value for this input. If specified, this is the initial value presented in the text box. If null or not specified, the initial value will be empty.**
<return value>	**String**	**If the user clicked the OK button, returns the contents of the text box. If the user clicked the cancel button, returns null.**

This static method provides a convenient means to prompt the user for arbitrary text input. No constraints are placed on user input.

input [S] org.carewebframework.ui.zk.PromptDialog

Parameter	Datatype	Description
message	**String**	**The text of the user prompt.**
title	**String**	**The title for the dialog.**
oldValue	**String**	**The prior value for this input. If specified, this is the initial value presented in the text box. If null or not specified, the initial value will be empty.**
itemNames	**List <String>**	**A list of item labels to be displayed in the list box. If null or has fewer entries than the `items` parameter, labels will be created for the remainder using the toString method of each item.**
items	**List <Object>**	**A list of objects that may be returned by the method. If null or has fewer entries than the `itemNames` parameter, the item label will be used for the remaining items.**
<return value>	**Object**	**If the user clicked the OK button, returns the object associated with the selected list item. If the user clicked the cancel button, returns null.**

This static method provides a convenient means to prompt the user for a list of choices, returning the object associated with the selected choice.

removeSavedResponse [S] org.carewebframework.ui.zk.PromptDialog

Parameter	Datatype	Description
responseId	**String**	**The unique id of the response to be removed.**

Removes the current user's stored response for the specified response id. If there is no response corresponding to this id, the request is ignored. After removing a stored response, the user will again be prompted when input is requested for this response id.

show [S] org.carewebframework.ui.zk.PromptDialog

Parameter	Datatype	Description
message	**String**	**The text of the user prompt.**
title	**String**	**The title for the dialog.**
buttonCaptions	**String**	**Button captions separated by vertical bars. A button will be created for each caption specified.**
styles (optional)	**String**	**SClass specifiers for icon and text, respectively, separated by vertical bars. May be null (the default).**
defaultButton (optional)	**String**	**The caption of the button that is to have the initial focus. May be null (the default).**
eventListener (optional)	**EventListener**	**An optional event listener to intercept and process click events. May be null (the default).**
responseId (optional)	**String**	**Uniquely identifies this response for purposes of saving and retrieving the last response. If not specified, null or empty, the response is not saved. Otherwise, if a saved response exists, it is returned without displaying the dialog. If a saved response does not exist, the user is prompted in the normal manner with the addition of a check box on the dialog asking if the response is to be saved. If this box is checked when the dialog is closed, the user's response is then saved as a user preference.**
excludedResponses (optional)	**String**	**Only applies if a `responseId` is specified. This is a list of responses that will not be saved and is specified in the same format as the `buttonCaptions` parameter.**
<return value>	**String**	**The caption of the button clicked by the user.**

This static method provides a convenient means to prompt the user for a number of response options presented as buttons.

showError [S] org.carewebframework.ui.zk.PromptDialog

Parameter	Datatype	Description
message	**String**	**The text of the user prompt.**
title (optional)	**String**	**The title for the dialog. If not specified, defaults to "Error".**

Displays an error dialog with the specified message and title.

showError [S] org.carewebframework.ui.zk.PromptDialog

Parameter	Datatype	Description
exception	**Throwable**	**The exception to display.**

Displays an error dialog for the specified exception.

showInfo [S] org.carewebframework.ui.zk.PromptDialog

Parameter	Datatype	Description
message	**String**	**The text of the user prompt.**
title (optional)	**String**	**The title for the dialog. If not specified, defaults to "Information".**

Displays an informational dialog with the specified message and title.

showText [S] org.carewebframework.ui.zk.PromptDialog

Parameter	Datatype	Description
message	**String**	**The text of the user prompt.**
title	**String**	**The title for the dialog.**

Displays an informational dialog with the specified message and title and using a fixed pitch font. This is useful for displaying information that has a columnated format, for example.

showWarning [S] org.carewebframework.ui.zk.PromptDialog

Parameter	Datatype	Description
message	**String**	**The text of the user prompt.**
title (optional)	**String**	**The title for the dialog. If not specified, defaults to "Warning".**

Displays a warning dialog with the specified message and title.

RowComparator Class

The **`RowComparator`** class provides a simple means to auto-wire data type-aware comparators to grid columns or list headers. It has the following static method:

autowireColumnComparators [S] org.carewebframework.ui.zk.RowComparator

Parameter	Datatype	Description
headers	**List <Object>**	**A list of `Column` or `Listheader` objects to be auto-wired.**

This static method auto-wires comparators to each of the **`Column`** or **`Listheader`** objects in the provided list. The method uses the id of each **`Column`** or **`Listheader`** to determine the property of the model object to be compared when sorting the grid rows or list items. The id may either be a property name or the name of the actual getter method (using standard bean conventions). For example, consider the following zul page snippet:

```
<listbox id="lstDocuments">
    <listhead>
        <listheader />
        <listheader label="Date"id="ReportDate" />
        <listheader label="Title" id="getReportTitle" />
        <listheader label="Signed"    id="isSigned" />
    </listhead>
</listbox>
```

And the following code to auto-wire the list headers:

```
RowComparator.autowireColumnComparators(lstDocuments.getListhead().getChildren());
```

The list box in the example declares four list headers. The first has no id and is, therefore, not auto-wired. The second has an id of **`ReportDate`**, so the **`RowComparator`** will infer a getter method name of **`getReportDate`**. The final two list headers have ids that start with standard getter method prefixes (**`get`**, **`is`**, and **`has`** are recognized). The **`RowComparator`** will assume these are the method names themselves. Because the **`RowComparator`** is data type-aware, it will correctly sort each of the three data types represented here – date, string, and Boolean, respectively.

TreeUtil Class

The **`TreeUtil`** class contains several static convenience methods for managing ZK's **`Tree`** component. For methods that specify a path, this consists of a backslash-delimited string of tree node labels that indicates the position of a tree node in the hierarchy, starting with the root node.

adjustVisibility [S] org.carewebframework.ui.zk.TreeUtil

Parameter	Datatype	Description
tree	Tree	A Tree component.

Hides the expand/collapse icons of tree nodes where there are no visible children.

adjustVisibility [S] org.carewebframework.ui.zk.TreeUtil

Parameter	Datatype	Description
parent	Treechildren	A Treechildren component.

Hides the expand/collapse icons of tree nodes where there are no visible children, beginning at the specified **`Treechildren`** component and recursing down the tree.

expand [S] org.carewebframework.ui.zk.TreeUtil

Parameter	Datatype	Description
tree	Tree	The tree whose nodes are to be expanded or collapsed.
depth	int	The depth to which nodes are to be visible (see discussion).

Provides a means to quickly expand tree nodes to a specified depth. A depth of 1 would collapse all tree nodes, leaving only first level nodes visible. A depth of 2 would expand all first level nodes while collapsing all others, making first and second level nodes visible.

expand [S] org.carewebframework.ui.zk.TreeUtil

Parameter	Datatype	Description
parent	Treechildren	`Treechildren` whose nodes are to be expanded or collapsed.
depth	int	The depth to which nodes are to be visible (see previous discussion).

This method is identical to its overloaded form above except that it accepts a **`Treechildren`** component instead of a **`Tree`**.

findNode [S] org.carewebframework.ui.zk.TreeUtil

Parameter	Datatype	Description
tree	Tree	The tree whose nodes are to be searched.
path	String	The path specifying the position of the tree node being sought.
create	boolean	If true, any tree nodes along the specified path that do not exist are created. If false, the search terminates when a matching node at any level along the path cannot be found.
clazz (optional)	Class <? extends Treeitem>	The class or subclass of `Treeitem` to create if node creation is allowed. Defaults to the class `Treeitem`.
caseSensitive (optional)	boolean	If true, node matching is case sensitive. The default is false.
<return value>	Treeitem	The tree item corresponding to the specified path. This may be null if node creation is not allowed and a matching node is not found.

Locates (and optionally creates) a tree item that corresponds to the specified path.

findNode [S] org.carewebframework.ui.zk.TreeUtil

Parameter	Datatype	Description
parent	Treechildren	The parent whose nodes are to be searched.
label	String	The label to find.
create	boolean	If true and a matching node is not found, one is created.
clazz	Class	The class or subclass of `Treeitem` to create if node creation is allowed.
caseSensitive	boolean	If true, node matching is case sensitive.
<return value>	Treeitem	The tree item whose label matches the specified label. This may be null if node creation is not allowed and a matching node is not found.

Locates (and optionally creates) a tree item whose label matches the specified value. Only the immediate children of the specified parent are searched.

getPath [S] org.carewebframework.ui.zk.TreeUtil

Parameter	Datatype	Description
item	Treeitem	The tree node whose path is to be returned.
<return value>	String	The path corresponding to the specified node. If the specified node is null, an empty string will be returned.

Returns the path that represents the position of the specified node in the tree hierarchy.

sort [S] org.carewebframework.ui.zk.TreeUtil

Parameter	Datatype	Description
tree	Tree	The tree whose nodes are to be sorted.

Sorts all levels of the specified tree. At each level, sibling nodes are arranged alphabetically (case-insensitive) by label.

sort [S] org.carewebframework.ui.zk.TreeUtil

Parameter	Datatype	Description
parent	Treechildren	The tree node whose children are to be sorted.
recurse	boolean	If true, sorting occurs at all levels below the specified parent. If false, sorting is limited to the immediate child nodes.

Sorts a tree beginning at the specified parent node. At each level, sibling nodes are arranged alphabetically (case-insensitive) by label. If recursion is allowed, all levels below the parent are sorted. Otherwise, only the level immediately below the parent node is sorted.

ZKUtil Class

The **`ZKUtil`** class provides several general-purpose static convenience methods for creating and manipulating ZK components.

addMask [S]

org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
component	**Component**	**The component to be disabled.**
caption	**String**	**The caption text to appear over the disabled component.**
popup	**Popup**	**A popup to display when context menu is invoked. Can be null for no popup.**

Places a semi-transparent mask over the specified component to disable user interaction.

addWatermark [S]

org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
component	**Component**	**The component to receive the watermark text.**
watermark	**String**	**The watermark text.**
color (optional)	**String**	**The color for watermark text in HTML format. Defaults to gray.**
font (optional)	**String**	**The font for watermark text. Defaults to the default web font.**
hideOnFocus (optional)	**boolean**	**If true, watermark text is hidden when component has focus. Defaults to false.**

Adds watermark (a.k.a., placeholder) text to the specified component. Will use native browser support if possible. Otherwise, uses the **`Text2ImageExtendlet`** class to simulate the effect.

createDialog [S]

org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
clazz	**Class <T extends Component>**	**The class of the dialog's top-level component.**
<return value>	**T**	**The top-level component of the class specified in the `clazz` parameter.**

Creates a dialog for the specified class. Auto-wires variables and event handlers and applies the **`MoveEventListener`** event listener to restrict movement of the dialog to the browser view port. This method makes several assumptions. First, the zul page backed by the class must have the same name as the class with a lowercase first character. Second, the zul page must be located in a subfolder under the **`web`** folder and named the same as the fully qualified package name. Third, the top-level component of the zul page must correspond to the specified class.

For example, if the class is **`org.carewebframework.sample.MyClass`**, the corresponding zul page must be **`web/org/carewebframework/sample/myClass.zul`** and the top-level component in the zul page must be of the type **`MyClass`**.

detach [S]

org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
components	List <? extends Component>	A list of components to detach.
exclusions (optional)	List <? extends Component>	A list of components to be excluded. Defaults to no exclusions.

Detaches a list of components with optional exclusions.

detachChildren [S]

org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
parent	Component	The component whose children are to be detached.
exclusions (optional)	List <? extends Component>	List of child components to be excluded. If null or not specified, all child components are detached.

Detaches some or all child components from the specified parent. The optional exclusions parameter may be used to exclude specific child components from the operation.

disableChildren [S]

org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
parent	Component	The component whose children are to be modified.
disable	boolean	The disable status.

Recurses over all descendants of the parent, setting the disable status of all components supporting ZK's `Disable` interface.

findAncestor [S]

org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
child	Component	The component whose ancestors will be searched.
ancestorClass	Class <T extends Component>	A `Component` subclass that is being sought.
<return value>	T	The first ancestor that is an instance of the specified class, or null if none found.

Finds the first ancestor of the specified child that is an instance of the specified class. This method will perform an upward breadth-first search of the component tree until it finds a component of the specified class or it reaches the root component of the tree.

findChild [S] org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
parent	Component	The component whose children will be searched.
childClass	Class <T extends Component>	A `Component` subclass that is being sought.
lastChild (optional)	Component	If specified, the search commences following this child component. If not specified, or null, the search commences with the first child of the parent component.
<return value>	T	The first child that is an instance of the specified class, or null if none found.

Finds the first child of the specified parent that is an instance of the specified class. This method searches only the immediate children of the parent component.

fireEvent [S] org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
event	Event	The event to be delivered.
eventListener (optional)	EventListener	The listener to receive the event. If not specified, the event will be delivered via `sendEvent`.

Fires the specified event, deferring delivery if a desktop is not currently active.

firstVisibleChild [S] org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
parent	Component	The component whose children are to be examined.
clazz (optional)	Class <T extends Component>	If specified, examine only components of this class. Defaults to the `Component` class.
recurse	boolean	If true, recurse over all descendants until a match is found.
<return value>	T	The first visible child encountered, or null if none found.

Returns the first visible child found, optionally recursing over all descendants.

formatExceptionForDisplay [S] org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
exception	Throwable	The exception to format.
<return value>	String	The exception text in displayable format, or null if exception parameter is null.

Formats an exception for display.

getAttribute [S]

org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
component	Component	The component containing the attribute map.
attrName	String	The name of the attribute.
clazz	Class <T>	The expected type of the attribute value.
<return value>	T	The value of the attribute, or null if not found or not of the specified type.

Returns the value of the named attribute from a component's attribute map.

getAttributeBoolean [S]

org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
component	Component	The component containing the attribute map.
attrName	String	The name of the attribute.
<return value>	boolean	The value of the attribute, or null if not found or not of the expected type.

Returns the value of the named attribute from a component's attribute map as a `Boolean`.

getAttributeComponent [S]

org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
component	Component	The component containing the attribute map.
attrName	String	The name of the attribute.
<return value>	Component	The value of the attribute, or null if not found or not of the expected type.

Returns the value of the named attribute from a component's attribute map as a `Component`.

getAttributeInt [S]

org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
component	Component	The component containing the attribute map.
attrName	String	The name of the attribute.
defaultVal	int	The default value to use if not found.
<return value>	int	The value of the attribute, or the default value if not found or not of the expected type.

Returns the value of the named attribute from a component's attribute map as an integer.

getAttributeList [S]

org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
component	Component	The component containing the attribute map.
attrName	String	The name of the attribute.
elementClass (optional)	Class	The expected type of the list's elements.
<return value>	List	The value of the attribute, or null if not found or not of the expected type.

Returns the value of the named attribute from a component's attribute map as a list.

getAttributeString [S]

org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
component	Component	The component containing the attribute map.
attrName	String	The name of the attribute.
<return value>	String	The value of the attribute, or null if not found or not of the expected type.

Returns the value of the named attribute from a component's attribute map as a string.

getAttributeXulElement [S]

org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
component	Component	The component containing the attribute map.
attrName	String	The name of the attribute.
<return value>	XulElement	The value of the attribute, or null if not found or not of the expected type.

Returns the value of the named attribute from a component's attribute map as a `XulElement`.

getEventOrigin [S]

org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
event	Event	The event whose origin is sought.
<return value>	Event	The origin of the specified event. If the event is not a forwarded event, returns the original event.

When a ZK event is forwarded, it is wrapped by a new event object of type `ForwardEvent` and it is that event that is forwarded. The forwarded event may itself be forwarded resulting in an arbitrarily long chain of forwarded events. Typically an event handler is interested only in the original event. This method provides a convenient means to ensure that the original event is being referenced.

getLabel [S] org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
key	String	The label key.
args (optional)	Object…	An optional argument list.
<return value>	String	The text value for the label, formatted with supplied arguments, or null if not found

Returns the text value given a label key.

getResourcePath [S] org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
pkg	Package	The package whose resource path is sought.
<return value>	String	The resource path corresponding to the specified package.

Returns the resource path associated with the specified package. This method uses the package name to create a file path and prefixes it with the special ZK `~./` syntax to indicate that it is a jar-embedded web resource. For example, if the specified package has a name of `org.carewebframework.xxxx`, this method would return a resource path string of `~./org/carewebframework/xxxx/`.

getResourcePath [S] org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
clazz	Class	The class whose resource path is sought.
<return value>	String	The resource path corresponding to the specified class.

This method makes a call to the overload of the `getResourcePath` function using the specified class' package (see above).

getResourcePath [S] org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
name	String	The name of the package whose resource path is sought.
<return value>	String	The resource path corresponding to the specified package name.

Returns the resource path associated with the given package (see above).

invokeJS [S] org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
functionName	String	The name of the JavaScript function to invoke.
depends	Component	The component on which the execution depends. If the component is removed before the execution request is sent, the request is ignored. May be null to specify no dependency.
args	Object…	An optional list of function arguments.

Invokes a JavaScript function on the client.

loadCachedPageDefinition [S]

org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
url	**String**	**The URL of the zul page whose cached page definition is sought.**
<return value>	**PageDefinition**	**The page definition of the specified zul page obtained from a global cache. If the page definition does not yet exist in the cache, it created from the specified zul page and placed in the cache.**

This method supports caching of the in-memory representation of a raw zul page (known as a page definition). When a zul page is processed by ZK, it is first converted to a page definition from which the page is actually constructed for display. Caching the definition speeds up processing of the associated zul page.

loadZulPage [S]

org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
url	**String**	**The URL of the zul page to be created.**
parent	**Component**	**The component that will become the parent of the created zul page. This may be null.**
args (optional)	**Map <Object, Object>**	**An optional map of parameters that may be referenced by the zul page.**
controller (optional)	**Object**	**An optional controller for the created zul page. If specified, components and events will be auto-wired to the controller.**
<return value>	**Component**	**Top-level component of the created zul page.**

This method loads a zul page for display. It has several overloads to support optional parameters.

loadZulPageDefinition [S]

org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
url	**String**	**The URL of the zul page whose definition is to be loaded.**
args (optional)	**Map**	**If specified, this map will be populated by any query parameters found in the URL.**
<return value>	**PageDefinition**	**The page definition of the specified URL.**

This method creates a **`PageDefinition`** from the specified zul page. If the **`args`** parameter is specified, the supplied map will be populated with any query parameters found in the URL.

moveChild [S]

org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
child	**Component**	**The child component to be moved.**
index	**int**	**The new position for this child.**

Moves a child component to the specified position relative to its siblings.

printToClient [S] org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
selectors	List <String>	A list of DOM selectors whose content will be printed.
styleSheets	List <String>	A list of style sheet refererences to be applied to the printed output.
preview	boolean	If true, open in preview mode. If false, submit directly for printing.

Prints portions of the DOM to the client via the browser's native printing capability. See the section on printing for more detail.

printToClient [S] org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
selector	String	A DOM selector whose content will be printed.
styleSheet	String	The url of a style sheet to be applied to printed output. May be null.
preview	boolean	If true, open in preview mode. If false, submit directly for printing.

Prints a portion of the DOM to the client via the browser's native printing capability. See the section on printing for more detail.

removeEventListeners [S] org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
component	Component	The component whose event listeners are to be removed.
eventName	String	The name of the event whose listeners are to be removed.
elClass (optional)	Class	If specified, only event listeners of this type will be removed.

Removes event listeners from a component.

removeMask [S] org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
component	Component	The component whose mask is to be removed.

Removes a mask from the specified component. See also **`addMask`**.

removeWatermark [S] org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
component	Component	The component whose watermark is to be removed.

Removes a watermark (a.k.a., placeholder) from the specified component. See also **`addWatermark`**.

suppressContextMenu [S]

org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
component	XulElement	The component whose context menu is to be suppressed.
noReplace (optional)	boolean	If true and a context menu exists for the specified component, do not replace it. If false (the default), an existing context menu is replaced.

Provides a means to suppress the display of the default browser context menu elicited when right-clicking on an HTML element that has not specified its own context menu. It sets the component's context property to a dummy popup component with no children and styled to be invisible.

swapChildren [S]

org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
child1	Component	First child to be swapped.
child2	Component	Second child to be swapped.

Swaps positions of two child components. Both components must have the same parent.

updateSclass [S]

org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
component	Component	Component whose sclass is to be changed.
className	String	The name of the style class.
remove	boolean	If true, the style class is removed. If false, the style class is appended to the sclass.
<return value>	String	The original sclass value.

Adds or removes a style class from a component's `sclass` property.

updateStyle [S]

org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
component	Component	The component whose style is to be modified.
styleName	String	The name of the style to modify.
styleValue	String	The new value for the style. If null or empty, the style is removed if it exists. Otherwise, if the style does not already exist, it is added with this value. If the style already exists, its value is replaced.
<return value>	String	The previous value for the style. Returns null if the style did not previously exist.

Provides a means to selectively add, remove, or modify a component's style without affecting other existing styles.

wireController [S] org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
component	**Component**	**The source component.**
controller (optional)	**Object**	**The controller to be wired. If not specified, the source component serves as its own controller.**

Wires instance variables and event handlers of a component's controller.

ZK2XML Class

This static utility class converts a ZK component tree into XML format. It is essentially the reverse process of converting a zul page into a component tree. It is presented as a debugging tool that aids in the visualization of an arbitrary component tree. There are two static methods available:

toDocument [S] org.carewebframework.ui.xml.ZK2XML

Parameter	Datatype	Description
root	**Component**	**The root of the component tree.**
excludedProperties (optional)	**String[]**	**An optional list of properties that should be excluded from the output. These may either be the property name (e.g., `uuid`) or a property name qualified by a component name (e.g., `window.uuid`). An entry may optionally be followed by "=" and a value to exclude matches with the specified value.**
<return value>	**Document**	**An XML document representing the component tree.**

Returns an XML document that represents the component tree starting at the specified root.

toXML [S] org.carewebframework.ui.xml.ZK2XML

Parameter	Datatype	Description
root	**Component**	**The root of the component tree.**
excludedProperties (optional)	**String[]**	**An optional list of properties that should be excluded from the output. These may either be the property name (e.g., `uuid`) or a property name qualified by a component name (e.g., `window.uuid`). An entry may optionally be followed by "=" and a value to exclude matches with the specified value.**
<return value>	**String**	**XML-formatted text representing the component tree.**

Returns an XML document in text form that represents the component tree starting at the specified root.

ZKDateUtil Class

This class provides some static convenience methods related to ZK's date and time input components.

getTime [S] org.carewebframework.ui.zk.ZKDateUtil

Parameter	Datatype	Description
datebox	**Datebox**	**A ZK date box component.**
timebox	**Timebox**	**A ZK time box component.**
<return value>	**Date**	**A date/time that combines the date from the date box component and the time from the time box component.**

Returns a date/time that combines the date specified by the date box component and the time specified by the time box component.

setTime [S] org.carewebframework.ui.zk.ZKDateUtil

Parameter	Datatype	Description
datebox	**Datebox**	**A ZK date box component.**
timebox	**Timebox**	**A ZK time box component.**
value	**Date**	**Sets the date part of the value in the date box component and the time part of the value in the time box component.**

Sets the date and time portions of date/time value into the respective date box and time box components.

ZK Component Extensions

We have created several ZK component extensions that add additional capabilities to existing ZK components or represent new components altogether. These are full-fledged language add-ons and can be referenced directly within a zul page by their respective tags.

Color Picker

class	org.carewebframework.ui.zk.ColorPicker
tag	colorpicker

This is a subclass of the ZK **`Bandbox`** component that supports the selection of a color from a palette of colors. This predated the ZK's **`Colorbox`** component and is presented as an alternative. In addition to the methods inherited from the **`Bandbox`** class, it has the following methods:

addColor

org.carewebframework.ui.zk.ColorPicker

Parameter	Datatype	Description
value	**String**	**The color RGB value in standard web format (e.g., "#228B22").**
name **(optional)**	**String**	**An optional display name for the color (e.g., "ForestGreen".**

Adds a color to the palette of color choices.

addColors

org.carewebframework.ui.zk.ColorPicker

Parameter	Datatype	Description
colors	**String[]**	**An array of RGB values.**

Adds an array of color RGB values to the palette of color choices.

addColors

org.carewebframework.ui.zk.ColorPicker

Parameter	Datatype	Description
colors	**String[][]**	**An array of color name/RGB value pairs.**

Adds an array of color name / RGB value pairs to the palette of color choices.

clear

org.carewebframework.ui.zk.ColorPicker

Removes all color choices from the palette.

getSelectedItem

org.carewebframework.ui.zk.ColorPicker

Parameter	Datatype	Description
<return value>	**Color**	**The selected color, or null if a color is not selected.**

Returns the value of the selected color.

getSelectedValue

org.carewebframework.ui.zk.ColorPicker

Parameter	Datatype	Description
<return value>	**String**	**The RGB value of the selected color, or null if a color is not selected.**

Returns the RGB value of the selected color.

getShowText

org.carewebframework.ui.zk.ColorPicker

Parameter	Datatype	Description
<return value>	boolean	The `ShowText` setting.

Returns the **`ShowText`** setting. If true, the color name (or RGB value if there is no color name) is displayed in the text box and the drop down button background is set to the color. If false, no text is displayed and the text box background is set to the color.

isAutoAdd

org.carewebframework.ui.zk.ColorPicker

Parameter	Datatype	Description
<return value>	boolean	The `AutoAdd` setting.

Returns the **`AutoAdd`** setting. If this setting is true, an attempt to set a color selection to one that is not in the current palette will cause it to be added automatically. If set to false, such an attempt will throw an exception.

reset

org.carewebframework.ui.zk.ColorPicker

Resets the color palette to the default choices.

setAutoAdd

org.carewebframework.ui.zk.ColorPicker

Parameter	Datatype	Description
autoAdd	boolean	The `AutoAdd` setting.

Sets the **`AutoAdd`** setting. If this setting is true, an attempt to set a color selection to one that is not in the current palette will cause it to be added automatically. If set to false, such an attempt will throw an exception.

setSelectedColor

org.carewebframework.ui.zk.ColorPicker

Parameter	Datatype	Description
value	String	RGB value for the color to be set.

Sets the currently selected color to the palette entry matching the specified RGB value.

setShowText

org.carewebframework.ui.zk.ColorPicker

Parameter	Datatype	Description
showText	boolean	The `ShowText` setting.

Sets the **`ShowText`** setting. If true, the color name (or RGB value if there is no color name) is displayed in the text box and the drop down button background is set to the color. If false, no text is displayed and the text box background is set to the color.

Date Box

class	org.carewebframework.ui.zk.DateboxEx
tag	datebox

This is a subclass of the ZK **`Datebox`** component that implements an enhanced date parsing algorithm supporting additional formats for date entry. It is otherwise identical to and replaces the ZK date box component. References to the ZK date box component in zul pages (via the **`datebox`** tag) will create instances of this class by default.

Date Range Chooser

class	org.carewebframework.ui.zk.DateRangeChooser
tag	datechooser

This is a subclass of the ZK **`Listbox`** component that supports the selection of pre-coordinated or custom date ranges. In addition to the methods inherited from the **`Listbox`** class, it has the following methods:

addChoice

org.carewebframework.ui.zk.DateRangeChooser

Parameter	Datatype	Description
range	**DateRange**	**A date range.**
isCustom	**boolean**	**If true, the range is a custom item and if another matching custom item exists, this range will not be added.**

Adds a date range to the choice list.

addChoice

org.carewebframework.ui.zk.DateRangeChooser

Parameter	Datatype	Description
range	**String**	**The serialized representation of a date range.**
isCustom	**boolean**	**If true, the range is a custom item and if another matching custom item exists, this range will not be added.**

Adds a date range, specified by its serialized string form, to the choice list.

clear

org.carewebframework.ui.zk.DateRangeChooser

Removes all items from the choice list. If the list item for adding custom choices is present, it will not be removed.

findMatchingItem

org.carewebframework.ui.zk.DateRangeChooser

Parameter	Datatype	Description
range	**DateRange**	**A date range.**
<return value>	**Listitem**	**The list item matching the specified date range.**

Locates and returns a list item with a date range matching the specified value. If no matching item is found, null is returned.

findMatchingItem

org.carewebframework.ui.zk.DateRangeChooser

Parameter	Datatype	Description
label	String	The label text to find.
<return value>	Listitem	The list item matching the specified label.

Locates and returns a list item with a label that matches the specified value (case-insensitive). If no matching item is found, null is returned.

getEndDate

org.carewebframework.ui.zk.DateRangeChooser

Parameter	Datatype	Description
<return value>	Date	The end date of the selected range, or null if no range is selected.

Returns the end date of the selected date range, or null if no date range is selected.

getListitems

org.carewebframework.ui.zk.DateRangeChooser

Parameter	Datatype	Description
<return value>	List <Listitem>	A list of all list items in the choice list.

Returns a list of all list items in the choice list.

getSelectedRange

org.carewebframework.ui.zk.DateRangeChooser

Parameter	Datatype	Description
<return value>	DateRange	The `DateRange` object corresponding to the current selection, or null if a range is not selected.

Returns the **`DateRange`** object corresponding to the selected date range, or null if no date range is selected.

getStartDate

org.carewebframework.ui.zk.DateRangeChooser

Parameter	Datatype	Description
<return value>	Date	The start date of the selected range, or null if no range is selected.

Returns the start date of the selected date range, or null if no date range is selected.

isAllowCustom

org.carewebframework.ui.zk.DateRangeChooser

Parameter	Datatype	Description
<return value>	boolean	True if custom date ranges are permitted.

Returns true if custom date ranges are permitted.

loadChoices

org.carewebframework.ui.zk.DateRangeChooser

Parameter	Datatype	Description
propertyName	String	The name of the property containing the choices. If null, loads the default list of choices.

Loads choices from the named property. If null, the default list of choices is loaded.

setAllowCustom

org.carewebframework.ui.zk.DateRangeChooser

Parameter	Datatype	Description
allowCustom	**boolean**	**If true, custom date ranges are permitted.**

Sets whether or not to allow custom date ranges. If set to true, a selectable list item will be included in the choice list that when clicked will prompt the user for a custom date range.

Date Range Picker

class	org.carewebframework.ui.zk.DateRangePicker
tag	datepicker

This is a subclass of the ZK **`Combobox`** component that supports the selection of pre-coordinated or custom date ranges. In addition to the methods inherited from the **`Combobox`** class, it has the following methods:

addChoice

org.carewebframework.ui.zk.DateRangePicker

Parameter	Datatype	Description
range	**DateRange**	**A date range.**
isCustom	**boolean**	**If true, the range is a custom item and if another matching custom item exists, this range will not be added.**

Adds a date range to the choice list.

addChoice

org.carewebframework.ui.zk.DateRangePicker

Parameter	Datatype	Description
range	**String**	**The serialized representation of a date range.**
isCustom	**boolean**	**If true, the range is a custom item and if another matching custom item exists, this range will not be added.**

Adds a date range, specified by its serialized string form, to the choice list.

clear

org.carewebframework.ui.zk.DateRangePicker

Removes all items from the choice list. If the list item for adding custom choices is present, it will not be removed.

findMatchingItem

org.carewebframework.ui.zk.DateRangePicker

Parameter	Datatype	Description
range	**DateRange**	**A date range.**
<return value>	**Comboitem**	**The item matching the specified date range.**

Locates and returns the item with a date range matching the specified value. If no matching item is found, null is returned.

findMatchingItem

org.carewebframework.ui.zk.DateRangePicker

Parameter	Datatype	Description
label	String	The label text to find.
<return value>	Comboitem	The item matching the specified label.

Locates and returns the item with a label that matches the specified value (case-insensitive). If no matching item is found, null is returned.

getEndDate

org.carewebframework.ui.zk.DateRangePicker

Parameter	Datatype	Description
<return value>	Date	The end date of the selected range, or null if no range is selected.

Returns the end date of the selected date range, or null if no date range is selected.

getComboitems

org.carewebframework.ui.zk.DateRangePicker

Parameter	Datatype	Description
<return value>	List<Comboitem>	A list of all items in the choice list.

Returns a list of all items in the choice list.

getPrompt

org.carewebframework.ui.zk.DateRangePicker

Parameter	Datatype	Description
<return value>	String	The prompt to display when there is no selection.

Returns the prompt to display when there is no selection.

getSelectedRange

org.carewebframework.ui.zk.DateRangePicker

Parameter	Datatype	Description
<return value>	DateRange	The `DateRange` object corresponding to the current selection, or null if a range is not selected.

Returns the **`DateRange`** object corresponding to the selected date range, or null if no date range is selected.

getStartDate

org.carewebframework.ui.zk.DateRangePicker

Parameter	Datatype	Description
<return value>	Date	The start date of the selected range, or null if no range is selected.

Returns the start date of the selected date range, or null if no date range is selected.

isAllowCustom

org.carewebframework.ui.zk.DateRangePicker

Parameter	Datatype	Description
<return value>	boolean	True if custom date ranges are permitted.

Returns true if custom date ranges are permitted.

loadChoices

org.carewebframework.ui.zk.DateRangePicker

Parameter	Datatype	Description
propertyName	**String**	**The name of the property containing the choices. If null, loads the default list of choices.**

Loads choices from the named property. If null, the default list of choices is loaded.

setAllowCustom

org.carewebframework.ui.zk.DateRangePicker

Parameter	Datatype	Description
allowCustom	**boolean**	**If true, custom date ranges are permitted.**

Sets whether or not to allow custom date ranges. If set to true, a selectable combo item will be included in the choice list that when clicked will prompt the user for a custom date range.

setPrompt

org.carewebframework.ui.zk.DateRangePicker

Parameter	Datatype	Description
prompt	**String**	**Sets the prompt to display when there is no selection.**

Sets the prompt to display when there is no selection.

Date Time Box

class	org.carewebframework.ui.zk.DateTimebox
tag	datetimebox

This is a subclass of the ZK **`Bandbox`** component that combines the selection of a date with a time component. Internally, it wraps a **`Datebox`** and a **`Timebox`** component. In addition to the methods inherited from the **`Bandbox`** class, it has the following methods:

getDate

org.carewebframework.ui.zk.DateTimebox

Parameter	Datatype	Description
<return value>	**Date**	**The date and time value.**

Returns the date and time value from the wrapped components.

getRequireTime

org.carewebframework.ui.zk.DateTimebox

Parameter	Datatype	Description
<return value>	**boolean**	**If true, a time must be entered.**

Returns true if the user is required to enter a time.

hasTime

org.carewebframework.ui.zk.DateTimebox

Parameter	Datatype	Description
<return value>	**boolean**	**True if a time value has been entered.**

Returns true if the user has entered a time.

setDate

org.carewebframework.ui.zk.DateTimebox

Parameter	Datatype	Description
date	Date	The date and time value.

Sets the date and time value for the wrapped components.

setRequireTime

org.carewebframework.ui.zk.DateTimebox

Parameter	Datatype	Description
requireTime	boolean	If true, a time must be entered.

Set to true if the user is required to enter a time.

setValue

org.carewebframework.ui.zk.DateTimebox

Parameter	Datatype	Description
value	String	A string value that is parsable to a date and time.

Sets the date of the component from a string value that is parsable to a date and time.

Selection Grid

class	org.carewebframework.ui.zk.SelectionGrid
tag	selectiongrid

This is a subclass of the ZK **`Grid`** component that adds the ability to select rows of a grid just as one would with a list box. Checkbox support is also provided. In addition to the methods inherited from the **`Grid`** class, it has the following methods:

addCheckbox

org.carewebframework.ui.zk.SelectionGrid

Parameter	Datatype	Description
row	Row	The row to receive the check box.

Adds a check box to the specified row. If the row already has an associated check box, the request is ignored.

addRow

org.carewebframework.ui.zk.SelectionGrid

Parameter	Datatype	Description
<return value>	Row	The newly added row.

Adds a new row to the end of the grid.

addRow

org.carewebframework.ui.zk.SelectionGrid

Parameter	Datatype	Description
index	int	The position for the new row. If < 0, appends the row to the end of the grid.
<return value>	Row	The newly added row.

Adds a new row to the grid at the specified position.

addRow

org.carewebframework.ui.zk.SelectionGrid

Parameter	Datatype	Description
row	**Row**	**The row to add.**
index	**int**	**The position for the row. If < 0, appends the row to the end of the grid.**
<return value>	**Row**	**The same as the `row` parameter.**

Adds a row to the grid at the specified position.

addRow

org.carewebframework.ui.zk.SelectionGrid

Parameter	Datatype	Description
row	**Row**	**The row to add.**
refRow	**Row**	**The row before which the added row will be inserted.**
<return value>	**Row**	**The same as the `row` parameter.**

Adds a row to the grid just before the specified reference row.

addRowToSelection

org.carewebframework.ui.zk.SelectionGrid

Parameter	Datatype	Description
row	**Row**	**The row to add.**

Adds a row to the list of selected rows.

clear

org.carewebframework.ui.zk.SelectionGrid

Removes all rows from the grid.

clearSelection

org.carewebframework.ui.zk.SelectionGrid

Deselects any selected rows.

fillRow

org.carewebframework.ui.zk.SelectionGrid

Parameter	Datatype	Description
row	**Row**	**The row to fill.**
colCount **(optional)**	**int**	**If specified, cells will be filled up to the specified column count. If not specified, all columns will be filled.**

Ensures that no cell in the given row is empty by adding an **`Hbox`** for any empty cell.

getAllRows

org.carewebframework.ui.zk.SelectionGrid

Parameter	Datatype	Description
<return value>	**List <Row>**	**A list of rows for this grid.**

Returns a list of all rows in the grid.

getRowAtIndex

org.carewebframework.ui.zk.SelectionGrid

Parameter	Datatype	Description
index	int	The index of desired row.
<return value>	Row	The row at the specified index.

Returns the row at the specified index.

getRowCount

org.carewebframework.ui.zk.SelectionGrid

Parameter	Datatype	Description
<return value>	int	The number of rows in this grid.

Returns the number of rows in the grid.

getSelectedCount

org.carewebframework.ui.zk.SelectionGrid

Parameter	Datatype	Description
<return value>	int	The number of selected rows in this grid.

Returns the number of selected rows in the grid.

isDisabled

org.carewebframework.ui.zk.SelectionGrid

Parameter	Datatype	Description
<return value>	boolean	True if the grid is disabled.

Returns the disabled state of the grid. A disabled grid will not respond to mouse clicks.

isDisabled

org.carewebframework.ui.zk.SelectionGrid

Parameter	Datatype	Description
row	Row	The row whose disabled status is sought.
<return value>	boolean	True if the row is disabled.

Returns the disabled state of the specified row. A disabled row will not respond to mouse clicks.

isSelected

org.carewebframework.ui.zk.SelectionGrid

Parameter	Datatype	Description
index	int	The index of the row.
<return value>	boolean	True if the specified row is selected.

Returns true if the row at the specified index is selected.

isSelected

org.carewebframework.ui.zk.SelectionGrid

Parameter	Datatype	Description
row	Row	The row.
<return value>	boolean	True if the specified row is selected.

Returns true if the specified row is selected.

isSelectOnClick

org.carewebframework.ui.zk.SelectionGrid

Parameter	Datatype	Description
<return value>	boolean	The select-on-click setting.

Returns the select-on-click setting. If this setting is set to true, clicking on a row is the same as clicking on its check box. If false, one must click the check box to select the row.

removeRow

org.carewebframework.ui.zk.SelectionGrid

Parameter	Datatype	Description
index	int	The index of the row to remove.

Removes the row at the specified index.

removeRow

org.carewebframework.ui.zk.SelectionGrid

Parameter	Datatype	Description
row	Row	The row to remove.

Removes the specified.

removeRowFromSelection

org.carewebframework.ui.zk.SelectionGrid

Parameter	Datatype	Description
row	Row	The row to remove.

Removes the specified row from the list of selected rows.

selectAll

org.carewebframework.ui.zk.SelectionGrid

Parameter	Datatype	Description
selected (optional)	boolean	The desired selection state. If not specified, defaults to true.

Sets the selection state for all rows to the specified value.

selectAllPage

org.carewebframework.ui.zk.SelectionGrid

Parameter	Datatype	Description
selected (optional)	boolean	The desired selection state. If not specified, defaults to true.

Sets the selection state for all rows on the current page to the specified value.

setDisabled

org.carewebframework.ui.zk.SelectionGrid

Parameter	Datatype	Description
value	boolean	True if the grid is to be disabled.

Sets the disabled state of the grid. A disabled grid will not respond to mouse clicks.

setDisabled org.carewebframework.ui.zk.SelectionGrid

Parameter	Datatype	Description
row	Row	The row whose disabled status is to be set.
value	boolean	True if the row is to be disabled.

Sets the disabled state of the specified row. A disabled row will not respond to mouse clicks.

setRowRenderer org.carewebframework.ui.zk.SelectionGrid

Parameter	Datatype	Description
renderer	RowRenderer	The row renderer for the grid. Must be an instance of `SelectionGridRenderer`.

Sets the row renderer for the grid. A run time exception is thrown if the renderer is not an instance of **`SelectionGridRenderer`**.

setSelected org.carewebframework.ui.zk.SelectionGrid

Parameter	Datatype	Description
index	int	The index of the row.
selected	boolean	True if the specified row is to be selected.

Set to true to select the row at the specified index.

setSelected org.carewebframework.ui.zk.SelectionGrid

Parameter	Datatype	Description
row	Row	The row.
selected	boolean	True if the specified row is to be selected.

Set to true to select the specified row.

setSelectOnClick org.carewebframework.ui.zk.SelectionGrid

Parameter	Datatype	Description
selectOnClick	boolean	The select-on-click setting.

Sets the select-on-click setting. If this setting is set to true, clicking on a row is the same as clicking on its check box. If false, one must click the check box to select the row.

Splitter Pane

class	org.carewebframework.ui.zk.SplitterPane
tag	splitterpane

This is a completely new component and represents a single pane within a splitter view. See the description of the splitter view for more information.

In addition to the methods inherited from the **`XulElement`** class, it has the following methods:

getAbsoluteSize

org.carewebframework.ui.zk.SplitterPane

Parameter	Datatype	Description
<return value>	int	The absolute size, in pixels, of the pane.

Returns the absolute size, in pixels, of the pane. For a horizontal orientation, this will be the pane's width; for vertical, the pane's height.

getRelativeSize

org.carewebframework.ui.zk.SplitterPane

Parameter	Datatype	Description
<return value>	double	The size of the pane relative to its parent, expressed as a percentage.

Returns the size of the pane relative to its parent, expressed as a percentage. For a horizontal orientation, this will be the pane's width; for vertical, the pane's height.

getTitle

org.carewebframework.ui.zk.SplitterPane

Parameter	Datatype	Description
<return value>	String	The text for the pane's title bar.

Returns the text of the pane's title bar.

isHorizontal

org.carewebframework.ui.zk.SplitterPane

Parameter	Datatype	Description
<return value>	boolean	True if the pane is in a horizontal orientation.

Returns true if the pane is in a horizontal orientation. This is a read-only property whose value is determined by the parent splitter view.

isRelative

org.carewebframework.ui.zk.SplitterPane

Parameter	Datatype	Description
<return value>	boolean	True if the pane's size is relative, false if fixed.

If true, the pane's size will remain relative to the size of its parent. When a pane with a relative size is restored during deserialization, its new size will be computed relative to the size of its parent. If false, the pane's size is fixed.

setAbsoluteSize

org.carewebframework.ui.zk.SplitterPane

Parameter	Datatype	Description
size	int	The size, in pixels, of the pane.

Sets the size, in pixels of this pane. For a horizontal orientation, this will be the pane's width; for vertical, the pane's height.

setRelativeSize

org.carewebframework.ui.zk.SplitterPane

Parameter	Datatype	Description
size	double	The size of the pane relative to its parent, expressed as a percentage.

Sets the size of the pane relative to its parent, expressed as a percentage. When a pane with a relative size is restored during deserialization, its new size will be computed relative to the size of its parent.

setTitle org.carewebframework.ui.zk.SplitterPane

Parameter	Datatype	Description
title	**String**	**The text for the pane's title bar.**

Sets the text of the pane's title bar.

Splitter View

class	org.carewebframework.ui.zk.SplitterView
tag	splitterview

This is completely new component, based on but not subclassed from the **`BorderLayout`** component. It provides the means to have a view with multiple panes separated by splitters in a horizontal or vertical orientation. It is offered as an alternative to an **`Hbox`** or **`Vbox`** component with splitters and does not suffer from some of the resizing anomalies seen with those components. And, unlike **`BorderLayout`**, it can contain any number of panels, though only in a single orientation (horizontal or vertical). An important feature of the splitter view is that it dynamically updates sizing information on the server side as this changes on the client.

In addition to the methods inherited from the **`XulElement`** class, it has the following methods:

SplitterView [C] org.carewebframework.ui.zk.SplitterView

Parameter	Datatype	Description
horizontal (optional)	**boolean**	**True if the splitter view will have a horizontal orientation; false if vertical. Defaults to true.**

Creates an instance of the splitter view with the specified (or default) orientation.

isHorizontal org.carewebframework.ui.zk.SplitterView

Parameter	Datatype	Description
<return value>	**boolean**	**True if the splitter view has a horizontal orientation; false if vertical.**

Returns true if the splitter view has a horizontal orientation (i.e., panes flow from left to right). Returns false if the orientation is vertical (i.e., panes flow from top to bottom).

setHorizontal org.carewebframework.ui.zk.SplitterView

Parameter	Datatype	Description
horizontal	**boolean**	**True if the splitter view will have a horizontal orientation; false if vertical.**

Set to true to give the splitter view a horizontal orientation (i.e., panes flow from left to right). Set to false if the orientation is to be vertical (i.e., panes flow from top to bottom).

Wonder Bar

The **`Wonderbar`** component exposes the JQuery UI autocompletion capability in a native ZK component. The component provides a drop down list of candidate selections as the user enters search text and, in the absence of search text, a list of default selections. The search function may be browser or server based.

Wonderbar Class

class	org.carewebframework.ui.zk.Wonderbar
tag	wonderbar

The principal component for the wonder bar is the **`Wonderbar`** class, a subclass of the **`InputElement`** ZK component. In addition to the standard methods inherited from its superclass, the **`Wonderbar`** class implements the following methods:

clear — org.carewebframework.ui.wonderbar.Wonderbar

Clears the input box and all selections.

close — org.carewebframework.ui.wonderbar.Wonderbar

Closes the drop down.

doSearch — org.carewebframework.ui.wonderbar.Wonderbar

Parameter	Datatype	Description
term	**String**	**The search text.**

Invokes a search for the specified text. Supports client- and server-side searches.

getChangeOnOKOnly — org.carewebframework.ui.wonderbar.Wonderbar

Parameter	Datatype	Description
<return value>	**boolean**	**The item selection behavior.**

If true, item selection occurs only when the **`Enter`** key is pressed. If false, item selection will also occur when the input box loses focus.

getClientMatchMode — org.carewebframework.ui.wonderbar.Wonderbar

Parameter	Datatype	Description
<return value>	**MatchMode**	**The match mode for client searches:** • **ANY_ORDER – Match any token order.** • **SAME_ORDER – Match only same token order (not necessarily contiguous).** • **ADJACENT – Match same token order and contiguous (not necessarily from start).** • **FROM_START – Match same token order, contiguous, and from start.**

Controls the matching behavior for client-side searches.

getClientThreshold

org.carewebframework.ui.wonderbar.Wonderbar

Parameter	Datatype	Description
<return value>	int	The threshold for transitioning from client-side to server-side search strategy.

For dual mode search providers, this property determines when to switch from client-side search strategy to server-side search strategy based on the number of searchable items.

getDefaultItems

org.carewebframework.ui.wonderbar.Wonderbar

Parameter	Datatype	Description
<return value>	WonderbarDefaults	The anchor component for default items.

Returns the anchor component for default items. May be null.

getItemRenderer

org.carewebframework.ui.wonderbar.Wonderbar

Parameter	Datatype	Description
<return value>	IWonderbarItemRenderer	The item renderer to use for rendering default and searchable items.

Returns the renderer implementation to use for rendering both default and searchable items.

getMaxSearchResults

org.carewebframework.ui.wonderbar.Wonderbar

Parameter	Datatype	Description
<return value>	int	The maximum number of search items to be returned by search provider.

Returns the maximum number of search items to be returned by the search provider. If the number of search items exceeds this threshold, a truncation indicator will appear after the last displayed search item.

getMinSearchCharacters

org.carewebframework.ui.wonderbar.Wonderbar

Parameter	Datatype	Description
<return value>	int	The minimum length of the search text before initiating a search.

Returns the minimum length of the search text before a search is initiated.

getOpenOnFocus

org.carewebframework.ui.wonderbar.Wonderbar

Parameter	Datatype	Description
<return value>	boolean	If true, the drop down list automatically appears when the component receives focus.

When true, the drop down list will automatically appear when the component receives focus. When false, the user must click the arrow button or initiate a search before the drop down appears.

getSearchProvider

org.carewebframework.ui.wonderbar.Wonderbar

Parameter	Datatype	Description
<return value>	IWonderbarSearchProvider	The search provider.

Returns the search provider implementation to use.

getSelectedData

org.carewebframework.ui.wonderbar.Wonderbar

Parameter	Datatype	Description
<return value>	Object	The data associated with the currently selected item, or null if no item selected.

Returns the data associated with the currently selected item, or null if no item is selected or the selected item has no associated data.

getSelectedItem

org.carewebframework.ui.wonderbar.Wonderbar

Parameter	Datatype	Description
<return value>	WonderbarItem	The currently selected item, or null if no item selected.

Returns the currently selected item, or null if no item has been selected.

isClient

org.carewebframework.ui.wonderbar.Wonderbar

Parameter	Datatype	Description
<return value>	boolean	True if the wonder bar is running in client search mode.

Returns true if the wonder bar is currently running in client search mode.

isSelectFirstItem

org.carewebframework.ui.wonderbar.Wonderbar

Parameter	Datatype	Description
<return value>	boolean	If true, the first selectable item will be selected by default.

If true, the first selectable item in the list will be selected by default. If false, no item is selected until explicitly selected by the user.

open

org.carewebframework.ui.wonderbar.Wonderbar

Drops down the selection list.

setChangeOnOKOnly

org.carewebframework.ui.wonderbar.Wonderbar

Parameter	Datatype	Description
value	boolean	Item selection behavior.

If true, item selection occurs only when the **`Enter`** key is pressed. If false, item selection will also occur when the input box loses focus.

setClientMatchMode

org.carewebframework.ui.wonderbar.Wonderbar

Parameter	Datatype	Description
value	MatchMode	The match mode for client searches. See `getClientMatchMode` for possible values.

Controls the matching behavior for client-side searches.

setClientThreshold

org.carewebframework.ui.wonderbar.Wonderbar

Parameter	Datatype	Description
value	int	Threshold for transitioning from client-side to server-side search strategy.

For dual mode search providers, this property determines when to switch from client-side search strategy to server-side search strategy based on the number of searchable items.

setItemRenderer

org.carewebframework.ui.wonderbar.Wonderbar

Parameter	Datatype	Description
value	IWonderbarItemRenderer	The item renderer to use for rendering default and searchable items.

Sets the renderer implementation to use for rendering both default and searchable items.

setMaxSearchResults

org.carewebframework.ui.wonderbar.Wonderbar

Parameter	Datatype	Description
value	int	The maximum number of search items to be returned by search provider.

Sets the maximum number of search items to be returned by the search provider. If the number of search items exceeds this threshold, a truncation indicator will appear after the last displayed search item.

setMinSearchCharacters

org.carewebframework.ui.wonderbar.Wonderbar

Parameter	Datatype	Description
value	int	The minimum length of the search text before initiating a search.

Sets the minimum length of the search text before a search is initiated.

setOpenOnFocus

org.carewebframework.ui.wonderbar.Wonderbar

Parameter	Datatype	Description
value	boolean	If true, the drop down list automatically appears when the component receives focus.

When true, the drop down list will automatically appear when the component receives focus. When false, the user must click the arrow button or initiate a search before the drop down appears.

setSearchProvider

org.carewebframework.ui.wonderbar.Wonderbar

Parameter	Datatype	Description
value	IWonderbarSearchProvider	The search provider.

Sets the search provider implementation to use.

setSelectedItem

org.carewebframework.ui.wonderbar.Wonderbar

Parameter	Datatype	Description
label	String	The label for the wonder bar item.
data	Object	The data associated with the wonder bar item. May be null.
fireEvent (optional)	boolean	If true, an `onWonderbarSelect` event is fired. If false, no event is fired. Defaults to true.

Creates a wonder bar item using the specified label and data and sets it as the current selection. Optionally, fires an **`onWonderbarSelect`** event.

setSelectedItem org.carewebframework.ui.wonderbar.Wonderbar

Parameter	Datatype	Description
value	**WonderbarItem**	**The item to be selected.**
fireEvent (optional)	**boolean**	**If true, a wonder bar select event is fired. If false, no event is fired. Defaults to true.**

Sets a wonder bar item as the current selection. Optionally, fires an **`onWonderbarSelect`** event.

WonderbarDefaults Class

class	org.carewebframework.ui.zk.WonderbarDefaults
tag	wonderbarDefaults

This is the anchor component for the default wonder bar items.

WonderbarItems Class

class	org.carewebframework.ui.zk.WonderbarItems
tag	wonderbarItems

This is the anchor component for wonder bar items generated by the search provider.

WonderbarItem Class

class	org.carewebframework.ui.zk.WonderbarItem
tag	wonderbarItem

An instance of this class represents a selectable item in the drop down selection list. It may be either a default item (parented by a **`WonderbarDefaults`** anchor component) or a search result (parented by a **`WonderbarItems`** anchor component).

It should be noted that a label value may include HTML markup by prefixing it with an **`<html>`** tag.

WonderbarItem [C] org.carewebframework.ui.wonderbar.WonderbarItem

Parameter	Datatype	Description
label (optional)	**String**	**The text label to be displayed.**
value (optional)	**String**	**The value to be set in the input box when the item is selected. If not specified, the label is used.**
data (optional)	**Object**	**Arbitrary data object to associate with the item.**

Creates an instance of a wonder bar item.

getCategory

org.carewebframework.ui.wonderbar.WonderbarItem

Parameter	Datatype	Description
<return value>	String	The item's category. May be null.

Returns the category to which this item belongs, if any. For the default renderer, the category appears to the right of the first item of consecutive items belonging to the same category.

getChoiceNumber

org.carewebframework.ui.wonderbar.WonderbarItem

Parameter	Datatype	Description
<return value>	int	The choice number for this item. Nonpositive integer values are ignored.

Returns the choice number for this item. When the user enters a positive integer value that matches this choice number, the associated item is highlighted.

getData

org.carewebframework.ui.wonderbar.WonderbarAbstractItem

Parameter	Datatype	Description
<return value>	Object	Arbitrary data associated with the item. May be null.

Returns the object instance associated with the wonder bar item.

getLabel

org.carewebframework.ui.wonderbar.WonderbarItem

Parameter	Datatype	Description
<return value>	String	The label value.

Returns the label value of the wonder bar item.

getSearchTerm

org.carewebframework.ui.wonderbar.WonderbarItem

Parameter	Datatype	Description
<return value>	String	The term to use for matching during searches. May be null.

Returns the term to be used for matching during a search. If not specified, the label value is used. For situations where the label value may not be suitable for searching (e.g., it contains HTML markup), this provides an alternative.

getUniqueKey

org.carewebframework.ui.wonderbar.WonderbarItem

Parameter	Datatype	Description
<return value>	String	A unique key. May be null.

Returns the unique key associated with this wonder bar item. This can be used to suppress the display of multiple items with the same unique key.

getUniquePriority

org.carewebframework.ui.wonderbar.WonderbarItem

Parameter	Datatype	Description
<return value>	int	Priority for unique key filtering.

Returns the priority for determining which of multiple items with the same unique key should be retained.

getValue

org.carewebframework.ui.wonderbar.WonderbarItem

Parameter	Datatype	Description
<return value>	String	The value to be placed in the input box when the item is selected. If null, the label is used.

Returns the value to be placed in the input box when the item is selected.

setCategory

org.carewebframework.ui.wonderbar.WonderbarItem

Parameter	Datatype	Description
value	String	The item's category. May be null.

Sets the category to which this item belongs, if any. For the default renderer, the category appears to the right of the first item of consecutive items belonging to the same category.

setChoiceNumber

org.carewebframework.ui.wonderbar.WonderbarItem

Parameter	Datatype	Description
value	int	The choice number for this item. Nonpositive integer values are ignored.

Sets the choice number for this item. When the user enters a positive integer value that matches this choice number, the associated item is highlighted.

setData

org.carewebframework.ui.wonderbar.WonderbarAbstractItem

Parameter	Datatype	Description
value	Object	Arbitrary data associated with the item. May be null.

Associates an arbitrary object instance with the wonder bar item.

setLabel

org.carewebframework.ui.wonderbar.WonderbarItem

Parameter	Datatype	Description
value	String	The label value.

Sets the label value of the wonder bar item.

setSearchTerm

org.carewebframework.ui.wonderbar.WonderbarItem

Parameter	Datatype	Description
value	String	The term to use for matching during searches. May be null.

Sets the term to be used for matching during a search. If not specified, the label value is used. For situations where the label value may not be suitable for searching (e.g., it contains HTML markup), this provides an alternative.

setUniqueKey

org.carewebframework.ui.wonderbar.WonderbarItem

Parameter	Datatype	Description
value	String	A unique key. May be null.

Sets the unique key associated with this wonder bar item. This can be used to suppress the display of multiple items with the same unique key.

setUniquePriority org.carewebframework.ui.wonderbar.WonderbarItem

Parameter	Datatype	Description
value	int	Priority for unique key filtering.

Sets the priority for determining which of multiple items with the same unique key should be retained.

setValue org.carewebframework.ui.wonderbar.WonderbarItem

Parameter	Datatype	Description
value	String	The value to be placed in the input box when the item is selected. If null, the label value is used.

Sets the value to be placed in the input box when the item is selected.

WonderbarGroup Class

class	org.carewebframework.ui.zk.WonderbarGroup
tag	wonderbarGroup

A wonder bar group item may appear alongside other wonder bar items. It is displayed as a non-selectable header for grouping similar items. It has the following methods:

WonderbarGroup [c] org.carewebframework.ui.wonderbar.WonderbarGroup

Parameter	Datatype	Description
label (optional)	String	The text label to be displayed.

Creates a wonder bar group item with the specified label.

getLabel org.carewebframework.ui.wonderbar.WonderbarGroup

Parameter	Datatype	Description
<return value>	String	The label value.

Returns the label value of the wonder bar group.

setLabel org.carewebframework.ui.wonderbar.WonderbarGroup

Parameter	Datatype	Description
value	String	The label value.

Sets the label value of the wonder bar group.

WonderbarSeparator Class

class	org.carewebframework.ui.zk.WonderbarSeparator
tag	wonderbarSeparator

Similar to a wonder bar group, the wonder bar separator may also appear alongside other wonder bar items. However, it has no label, but rather provides visual separation between items.

IWonderbarClientSearchProvider Interface

A search provider may be client-based, server-based or both. A client-based search provider must implement the **`IWonderbarClientSearchProvider`** interface. Unlike a server-based search provider, the client-based search provider has no inherent search capability. Rather, it provides all of its searchable content to the client where the search logic resides. This minimizes network traffic for performing search operations, but scales poorly for large search content volumes where a server-based approach is more suitable.

The **`IWonderbarClientSearchProvider`** interface declares the following methods:

getDefaultItems org.carewebframework.ui.wonderbar.IWonderbarClientSearchProvider

Parameter	Datatype	Description
<return value>	List	The default wonder bar items. May be null.

Returns a list of the default wonder bar items.

getAllItems org.carewebframework.ui.wonderbar.IWonderbarClientSearchProvider

Parameter	Datatype	Description
<return value>	List	The searchable wonder bar items. May be null.

Returns a list of all searchable wonder bar items.

IWonderbarServerSearchProvider Interface

Unlike a client-based search provider, a server-based search provider must provide its own search logic. It receives a search request from the client, executes its search logic against the searchable content, and delivers the search hits back to the client for display. This incurs more network I/O for a given search operation than does the client-based approach, but scales better for large search content volumes.

Every server-based search provider must implement the **`IWonderbarServerSearchProvider`** interface, which declares the following methods:

getDefaultItems org.carewebframework.ui.wonderbar.IWonderbarServerSearchProvider

Parameter	Datatype	Description
<return value>	List	The default wonder bar items. May be null.

Returns a list of the default wonder bar items.

getSearchResults org.carewebframework.ui.wonderbar.IWonderbarServerSearchProvider

Parameter	Datatype	Description
search	String	The search text.
maxItems	int	The maximum number of items to return. The implementation may use this for performance reasons. The returned list will be truncated if it exceeds this maximum even if the provider returns more.
hits	List	The list to receive the matched items.
<return value>	boolean	True if all matched items were returned. False if the list was truncated.

Returns a list of search results.

WonderbarSelectEvent Class

This is the event object generated when a wonder bar item is selected. It exposes the following methods:

getKeys org.carewebframework.ui.wonderbar.WonderbarSelectEvent

Parameter	Datatype	Description
<return value>	int	The keypress state at the time of the selection.

Returns a bit map that reflects the key press state at the time of the selection. See ZK's documentation of the **`MouseEvent`** class for more information.

getSelectedItem org.carewebframework.ui.wonderbar.WonderbarSelectEvent

Parameter	Datatype	Description
<return value>	WonderbarItem	The selected wonder bar item.

Returns the wonder bar item that was selected.

WonderbarUtil Class

This is a utility class of static utility methods. It has the following methods.

addDefaultItems org.carewebframework.ui.wonderbar.WonderbarUtil

Parameter	Datatype	Description
wonderbar	Wonderbar	The wonder bar to receive default items.
items	Collection <WonderbarAbstractItem>	List of default items to be added.

Convenience method for adding default items to a wonder bar. Automatically creates the anchor component, **`WonderbarDefaults`**, if necessary.

matches

org.carewebframework.ui.wonderbar.WonderbarUtil

Parameter	Datatype	Description
patterns	List <String>	A list of search pattern tokens to be matched.
value	String	The search text.
mode	MatchMode	The match mode. One of: • ANY_ORDER – Match any token order. • SAME_ORDER – Match only same token order (not necessarily contiguous). • ADJACENT – Match same token order and contiguous (not necessarily from start). • FROM_START – Match same token order, contiguous, and from start.
<return value>	boolean	True if the value is a match.

Returns true if the value matches the specified pattern set. This method is more efficient when comparing multiple values against the same pattern set.

matches

org.carewebframework.ui.wonderbar.WonderbarUtil

Parameter	Datatype	Description
pattern	String	A search pattern to be matched.
value	String	The search text.
mode	MatchMode	The match mode. One of: • ANY_ORDER – Match any token order. • SAME_ORDER – Match only same token order (not necessarily contiguous). • ADJACENT – Match same token order and contiguous (not necessarily from start). • FROM_START – Match same token order, contiguous, and from start.
<return value>	boolean	True if the value matches the search pattern.

Returns true if the value matches the specified pattern.

tokenize

org.carewebframework.ui.wonderbar.WonderbarUtil

Parameter	Datatype	Description
text	String	The text to be tokenized.
sortByLength (optional)	boolean	If true, tokens are sorted by length, the longest first. Defaults to false.
<return value>	List <String>	A list of string tokens.

Breaks the text into individual tokens at word boundaries. Trims each token, converts to lower case, and removes empty tokens.

CWF Component Model

The CWF employs an extensible component model that supports a dynamic and configurable layout of UI elements. All components descend from a common abstract base class, **`UIElementBase`**. Each has methods that describe its capabilities, support its visual rendering, and handle serialization and deserialization of its state. These components do not directly subclass ZK's visual components, but do use ZK components in their rendering logic. In a later section, we describe how to create your own custom UI elements.

The CWF supplies a number of UI elements out of the box. These support a variety of UI layouts such as tab views, pane views, step-oriented views, etc. Some UI elements may serve as containers for other UI elements (such as a tab pane). While all descend from a common abstract class **`UIElementBase`**, the majority of UI elements directly descend from an abstract subclass of **`UIElementBase`** called **`UIElementZKBase`**. These and related classes are documented in detail below followed by an overview of available derived classes.

PluginDefinition Class

Every concrete UI element class has an associated plugin definition that provides metadata about the element. The definition's declaration may be found in the Spring configuration file that is packaged with UI element. From this declaration, an instance of the **`PluginDefinition`** class is created and registered with a plugin registry. The plugin definition provides essential information to the CWF such as how to serialize and deserialize the UI element and how to manipulate it within the Layout Designer. The following methods of the **`PluginDefinition`** class will be of potential interest to the programmer:

createElement — org.carewebframework.shell.plugins.PluginDefinition

Parameter	Datatype	Description
parent	**UIElementBase**	**The UI element that will serve as the parent for the newly created UI element. This may be null.**
propertySource	**IPropertyProvider**	**A source of property values that will be used for initializing properties of the newly created UI element. This may be null.**
<return value>	**UIElementBase**	**The newly created UI element.**

Creates an instance of a UI element based on its definition. If a property source is specified, this is used to initialize properties on the newly created element.

getCategory — org.carewebframework.shell.plugins.PluginDefinition

Parameter	Datatype	Description
<return value>	**String**	**The tree node path that categorizes the associated UI element.**

Returns the category under which the UI element is to be classified. This is specified as a tree node path using backslash characters as delimiters. This setting affects

where the Layout Designer displays the UI element in tree-based views of available UI elements.

getClazz

org.carewebframework.shell.plugins.PluginDefinition

Parameter	Datatype	Description
<return value>	Class <? extends UIElementBase>	The class that implements the associated UI element.

Returns the class that implements the associated UI element. The CWF uses this information to create instances of the UI element.

getCopyright

org.carewebframework.shell.plugins.PluginDefinition

Parameter	Datatype	Description
<return value>	String	Any copyright information for the UI element.

Returns any copyright information applicable to the UI element. The Layout Designer displays this information when the about box for the UI element is invoked.

getCreator

org.carewebframework.shell.plugins.PluginDefinition

Parameter	Datatype	Description
<return value>	String	The name of the creator of the UI element.

Returns the name of the creator of the UI element. The Layout Designer displays this information when the about box for the UI element is invoked.

getDefinition [S]

org.carewebframework.shell.plugins.PluginDefinition

Parameter	Datatype	Description
clazz	Class <? extends UIElementBase>	The UI element class whose associated plugin definition is sought.
<return value>	PluginDefinition	A plugin definition, or null if not found.

Returns the plugin definition associated with the specified UI element class.

getDefinition [S]

org.carewebframework.shell.plugins.PluginDefinition

Parameter	Datatype	Description
tag	String	The tag of a UI element whose associated plugin definition is sought.
<return value>	PluginDefinition	A plugin definition, or null if not found.

Returns the plugin definition given the XML tag name of a UI element.

getDescription

org.carewebframework.shell.plugins.PluginDefinition

Parameter	Datatype	Description
<return value>	String	A description of the UI element's function.

Returns a description of the UI element's function. The Layout Designer displays this information when the about box for the UI element is invoked.

getIcon

org.carewebframework.shell.plugins.PluginDefinition

Parameter	Datatype	Description
<return value>	String	A URL that references an icon to be associated with this UI element.

Returns the URL that references an icon to be associated with this UI element. This information may be used by the Layout Designer to display an icon representative of the UI element.

getId

org.carewebframework.shell.plugins.PluginDefinition

Parameter	Datatype	Description
<return value>	String	The unique id associated with the UI element.

Returns the unique id associated with the UI element. This id is used as a tag when serializing the UI element to XML.

getName

org.carewebframework.shell.plugins.PluginDefinition

Parameter	Datatype	Description
<return value>	String	The display name for the UI element.

Returns the display name for the UI element. The Layout Designer uses this name to provide a user-friendly representation of the UI element.

getAuthorities

org.carewebframework.shell.plugins.PluginDefinition

Parameter	Datatype	Description
<return value>	List <Authority>	Returns a list of authorities required to access this UI element.

Returns a list of authorities required to access the UI element. This list is never null, but may be empty indicating no special authorities are required. The CWF will prevent the instantiation of any UI element for which the user does not have the requisite authorities. See the **`isRequiresAll`** method for information on how multiple entries in this list are interpreted.

getProperties

org.carewebframework.shell.plugins.PluginDefinition

Parameter	Datatype	Description
<return value>	List <PropertyInfo>	A list of property descriptors associated with this UI element.

Returns a list of property descriptors associated with the UI element. This information is used when serializing and deserializing a UI element and by the Layout Designer when presenting a property editor to the user.

getReleased

org.carewebframework.shell.plugins.PluginDefinition

Parameter	Datatype	Description
<return value>	String	Release information about the UI element. This is typically a release date.

Returns release information pertaining to the UI element. The Layout Designer displays this information when the about box for the UI element is invoked.

getResources

org.carewebframework.shell.plugins.PluginDefinition

Parameter	Datatype	Description
clazz (optional)	**Class <PluginResource>**	**If specified, restricts the returned values to those belonging to the specified `PluginResource` subclass.**
<return value>	**List <PluginResource>**	**A list of resources associated with the UI element, optionally restricted by the `clazz` parameter.**

This returns resources associated with the UI element (see the discussion of CWF plugin development for a list of resource types).

getSource

org.carewebframework.shell.plugins.PluginDefinition

Parameter	Datatype	Description
<return value>	**String**	**The source vendor supplying the UI element.**

Returns the source vendor supplying the UI element. The Layout Designer displays this information when the about box for the UI element is invoked.

getUrl

org.carewebframework.shell.plugins.PluginDefinition

Parameter	Datatype	Description
<return value>	**String**	**The URL of the principal zul page for the UI element.**

Returns the URL of the principal zul page for the UI element.

getVersion

org.carewebframework.shell.plugins.PluginDefinition

Parameter	Datatype	Description
<return value>	**String**	**Version information for the UI element.**

Returns any version information applicable to the UI element. The Layout Designer displays this information when the about box for the UI element is invoked.

hasEditableProperties

org.carewebframework.shell.plugins.PluginDefinition

Parameter	Datatype	Description
<return value>	**boolean**	**True if the UI element has any editable properties.**

Returns true if the UI element has any editable properties.

isDisabled

org.carewebframework.shell.plugins.PluginDefinition

Parameter	Datatype	Description
<return value>	**boolean**	**True if the UI element has been disabled.**

Returns true if the UI element is disabled. The CWF will refuse to instantiate a UI element that has been disabled.

isForbidden

org.carewebframework.shell.plugins.PluginDefinition

Parameter	Datatype	Description
<return value>	boolean	True if access to the UI element is not permitted.

Returns true if access to the UI element is not permitted for the requesting user. This function evaluates the user's security authorities against those required for access to the UI element (see **`getAuthorities`** and **`isRequiresAll`**).

isInternal

org.carewebframework.shell.plugins.PluginDefinition

Parameter	Datatype	Description
<return value>	boolean	Returns true if this definition represents an internal UI element.

Returns true if this definition represents an internal UI element. An internal UI element is one that is not created directly by the deserializer.

isLazyLoad

org.carewebframework.shell.plugins.PluginDefinition

Parameter	Datatype	Description
<return value>	boolean	If true, internal components wrapped by this UI element may not be instantiated until the UI element is first activated.

If true, some or all internal components wrapped by this UI element may not be instantiated until the UI element is first activated. The implementation of the UI element may choose to ignore this setting. This setting is provided to allow for potential performance optimizations when creating the initial layout.

isRequiresAll

org.carewebframework.shell.plugins.PluginDefinition

Parameter	Datatype	Description
<return value>	boolean	If true, the user must possess all authorities associated with the UI element to access it. If false, possessing any one of the associated authorities is sufficient.

This setting determines how multiple authorities associated with a UI element are interpreted. If no authorities or only one authority are associated with a UI element, this setting has no effect. Otherwise, when true, the user must possess all associated authorities in order to access (instantiate) the associated UI element. If false, possessing any one of the associated authorities is sufficient to be granted access.

UIElementBase Class

This is the abstract base class for all UI elements in the CWF, though most descend from its immediate descendant, **`UIElementZKBase`**.

about

org.carewebframework.shell.layout.UIElementBase

Invokes the about dialog for the associated UI element. A default implementation is provided, but this may be overridden if necessary to display a custom about dialog.

activate

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
activate	boolean	If true, the component is activated. If false, it is deactivated.

Activates or inactivates a UI element. In general, this method should not be overridden to introduce new behavior. Rather, if a UI element must change its visual state in response to a change in activation state, it should override the **`updateVisibility`** method. If a UI element requires special activation logic for its children (e.g., if it allows only one child to be active at a time), it should override the **`activateChildren`** method.

activateChildren

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
activate	boolean	If true, child elements are activated. If false, they are deactivated.

Activates or inactivates the children of a UI element. The default implementation calls the **`activate`** method on each child UI element. If a UI element requires special activation logic for its children (e.g., if it allows only one child to be active at a time), it should override this method.

addChild

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
child	UIElementBase	The child UI element to be added.
doEvent (optional)	boolean	If true, the `beforeAddChild` and `afterAddChild` methods are called. Defaults to true.

Adds a child UI element to this UI element. A runtime exception may be thrown if this UI element is not a container or if the child element is not compatible (see **`canAcceptChild`** and **`canAcceptParent`** methods). If **`doEvent`** is true, the **`beforeAddChild`** and **`afterAddChild`** methods are called before and after (respectively) adding the child.

afterAddChild

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
child	UIElementBase	The child UI element added.

Called immediately after adding a child UI element. The default implementation does nothing. Override to introduce any special behavior after a child UI element is added.

afterInitialize

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
deserializing	boolean	If true, initialization is being requested during deserialization.

Subclasses may override this method to implement any additional operations that are necessary after this UI element is initialized (i.e., after property values and the parent element are set).

afterMoveTo

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
index	int	New position for the element.

Called after an element has been moved to a different position relative to its siblings. Subclasses may override to make any necessary adjustments to wrapped components.

afterParentChanged

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
oldParent	UIElementBase	The previous parent.

Called after the parent has been changed.

afterRemoveChild

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
child	UIElementBase	The child UI element just removed.

Called after a child is logically removed from its parent.

applyColor

org.carewebframework.shell.layout.UIElementBase

Provides a default implementation for applying the current color setting to the wrapped components, which calls the parameterized **`applyColor`** method, passing the outer and inner wrapped components. Override to modify this default behavior.

applyColor

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
component	Object	The wrapped component whose color is to be changed.

Applies the current color setting to the specified component. The default implementation does nothing. Override to provide the necessary logic.

applyHint

org.carewebframework.shell.layout.UIElementBase

Provides a default implementation for setting the hint text of wrapped components, which calls the parameterized **`applyHint`** method, passing the outer and inner wrapped components. Override to modify this default behavior.

applyHint

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
component	Object	The wrapped component whose hint text is to be changed.

Applies the hint text setting to the specified component. The default implementation does nothing. Override to provide the necessary logic.

beforeAddChild

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
child	**UIElementBase**	**The child UI element to be added.**

Called immediately before adding a child UI element. The default implementation does nothing. Override to introduce any special behavior before a child UI element is added.

beforeInitialize

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
deserializing	**boolean**	**If true, initialization is being requested during deserialization.**

Subclasses may override this method to implement any additional operations that are necessary before this UI element is initialized (i.e., before property values and the parent element are set).

beforeParentChanged

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
newParent	**UIElementBase**	**The new parent.**

Called before the parent has been changed.

beforeRemoveChild

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
child	**UIElementBase**	**The child UI element to be removed.**

Called before a child is logically removed from its parent.

bind

org.carewebframework.shell.layout.UIElementBase

Override to bind wrapped components to the UI.

bringToFront

org.carewebframework.shell.layout.UIElementBase

Causes the UI element to be brought to the forefront of the UI. The default implementation simply calls the same method on the parent UI element. A subclass should override to provide any additional logic necessary to ensure that the UI element is visible to the user.

canAcceptChild

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
<return value>	**boolean**	**If true, the child may be accepted by this UI element. If false, the `RejectReason` property will be set to provide a displayable reason why the child was rejected.**

Determines whether this UI element can accept a child. If the method returns false, the **`getRejectReason`** method will return a displayable reason. The default implementation will reject a child if this UI element is not a container or if adding a child would exceed the maximum number of allowable children. Note that this method merely determines if the UI element can accept a child without regard to the

child's class. Thus, an attempt to add a specific child may still be rejected even if this method returns true. Override this method to provide any additional constraints.

canAcceptChild org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
child	UIElementBase	The child UI element to be added.
<return value>	boolean	If true, the child may be accepted by this UI element. If false, the `RejectReason` property will be set to provide a displayable reason why the child was rejected.

Determines whether this UI element can accept the specified child. If the child is rejected, the **`getRejectReason`** method will return a displayable reason. The default implementation will reject a child if this UI element is not a container, if adding the child would exceed the maximum number of allowable children, or if the child's class is not registered as a valid child class for this UI element. Override this method to provide any additional constraints.

canAcceptChild org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
clazz	Class <? extends UIElementBase>	The class of the child UI element to be added.
<return value>	boolean	If true, the child may be accepted by this UI element. If false, the `RejectReason` property will be set to provide a displayable reason why the child was rejected.

Determines whether this UI element can accept the specified child. If the child is rejected, the **`getRejectReason`** method will return a displayable reason. The default implementation will reject a child if this UI element is not a container, if adding the child would exceed the maximum number of allowable children, or if the child's class is not registered as a valid child class for this UI element. Override this method to provide any additional constraints.

canAcceptChild [S] org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
parentClass	Class <? extends UIElementBase>	The class of the parent UI element.
childClass	Class <? extends UIElementBase>	The class of the child UI element.
<return value>	boolean	True if the parent class can serve as a parent for the child class.

This static method determines whether one **`UIElementBase`**-derived class can serve as a child of another. It uses information registered using the **`registerAllowedChildClass`** static method to make this determination.

canAcceptParent org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
<return value>	boolean	If true, the parent may be accepted by this UI element. If false, the `RejectReason` property will be set to provide a displayable reason why the parent was rejected.

Determines whether this UI element can accept a parent. If the method returns false, the **`getRejectReason`** method will return a displayable reason. The default implementation will reject a parent if this UI element has no registered classes that may serve as a parent (this is normally only true for the **`UIElementDesktop`** class which serves as the top level class for all layouts). Note that this method merely determines if the UI element can accept a parent without regard to the parent's class. Thus, an attempt to add a specific parent may still be rejected even if this method returns true. Override this method to provide any additional constraints.

canAcceptParent org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
parent	UIElementBase	The parent UI element to be added.
<return value>	boolean	If true, the parent may be accepted by this UI element. If false, the `RejectReason` property will be set to provide a displayable reason why the parent was rejected.

Determines whether this UI element can accept the specified parent. If the parent is rejected, the **`getRejectReason`** method will return a displayable reason. The default implementation will reject a parent if the parent's class is not registered as a valid parent class for this UI element. Override this method to provide any additional constraints.

canAcceptParent org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
clazz	Class <? extends UIElementBase>	The class of the parent UI element to be added.
<return value>	boolean	If true, the parent may be accepted by this UI element. If false, the `RejectReason` property will be set to provide a displayable reason why the parent was rejected.

Determines whether this UI element can accept the specified parent class. If the parent class is rejected, the **`getRejectReason`** method will return a displayable reason. The default implementation will reject a parent if the parent's class is not registered as a valid parent class for this UI element. Override this method to provide any additional constraints.

canAcceptParent [S]

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
childClass	Class <? extends UIElementBase>	The class of the child UI element.
parentClass	Class <? extends UIElementBase>	The class of the parent UI element.
<return value>	boolean	True if the parent class can serve as a parent for the child class.

This static method determines whether one **`UIElementBase`**-derived class can serve as a parent to another. It uses information registered using the **`registerAllowedParentClass`** static method to make this determination.

destroy

org.carewebframework.shell.layout.UIElementBase

Called when the UI element is about to be destroyed. The default implementation does nothing, but can be overridden to perform any special cleanup that may be required.

editProperties

org.carewebframework.shell.layout.UIElementBase

Displays a property editor for the UI element. The default implementation simply displays an information dialog.

findChildElement

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
clazz	Class <T extends UIElementBase>	The class of the child UI element.
<return value>	T	A child UI element of the specified class, or null if one was not found.

Searches the UI element's children for a child of the specified class. If no element of the specified class is found, the children's children are searched until a matching element is found or the end of the subtree has been reached.

getAncestor

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
clazz	Class <T extends UIElementBase>	The class of the ancestor UI element.
<return value>	T	An ancestor UI element of the specified class, or null if one was not found.

Searches the UI element's parent chain for an ancestor of the specified class.

getChild

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
clazz	Class <T extends UIElementBase>	The class of the child UI element.
last	UIElementBase	The child UI element after which to begin the search. If null, the search begins with the first child.
<return value>	T	A child UI element of the specified class, or null if one was not found.

Searches the immediate children for a UI element of the specified class. If `last` is specified, the search begins after that child. Otherwise, the search begins with the first child. If no matching child is found, returns null.

getChild

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
index	int	The index of the child UI element.
<return value>	UIElementBase	The child UI element at the specified index.

Returns the child UI element at the specified index.

getChildCount

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
<return value>	int	The number of children for this UI element.

Returns the number of children for this UI element.

getChildren

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
<return value>	Iterable <UIElementBase>	An iterable list of this UI element's children.

Returns an iterable list of this UI element's children.

getColor

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
<return value>	String	Color value as an HTML-formatted string.

Returns the color setting as an HTML-formatted string. May be null.

getDefinition

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
<return value>	PluginDefinition	The plugin definition for this UI element.

Returns the plugin definition that describes this UI element.

getDisplayName

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
<return value>	String	The display name for this UI element.

Returns the display name for this UI element. The default implementation returns the name supplied by the associated plugin definition. Subclasses may override to supply a different display name.

getFirstChild

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
<return value>	UIElementBase	The first child, or null if no children.

Returns the first child element, or null if one does not exist.

getFirstVisibleChild

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
<return value>	UIElementBase	The first visible child, or null if no visible children.

Returns the first visible child element, or null if one does not exist.

getHint

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
<return value>	String	The hint text.

Returns the hint text for this UI element.

getIndex

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
<return value>	int	This UI element's index in its parent's list of children. If there is no parent, returns -1.

Returns the index of this UI element in its parent's list of children, or -1 if there is no parent.

getInnerComponent

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
<return value>	Object	The innermost wrapped UI component.

Returns the innermost wrapped UI component. This will typically be a ZK component. This would be the component that would be attached to the outer component of a child UI element. For many UI elements, the inner and outer components are the same.

getInstanceName

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
<return value>	String	The instance name for this UI element.

Returns the instance name for this UI element. By default, this is the same as the display name, but subclasses may override to provide additional information that would distinguish multiple instances of the same UI element.

getLastChild

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
<return value>	UIElementBase	The last child UI element, or null if no children exist.

Returns the last child UI element, or null if one does not exist.

getLastVisibleChild

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
<return value>	UIElementBase	The last visible child UI element, or null if no visible children exist.

Returns the last visible child UI element, or null if one does not exist.

getNextSibling

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
visibleOnly	boolean	If true, ignore all non-visible siblings.
<return value>	UIElementBase	The sibling following this one that matches the visibility criterion, or null if none found.

Returns the first sibling following this one and matching the specified visibility criterion, or null if one does not exist.

getOuterComponent

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
<return value>	Object	The outermost wrapped UI component.

Returns the outermost wrapped UI component. This will typically be a ZK component. This would be the component that would be attached to the inner component of the parent UI element. For many UI elements, the inner and outer components are the same.

getParent

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
<return value>	UIElementBase	The parent of this UI element. May be null.

Returns the parent of this UI element, or null if one does not exist.

getPropEditClass

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
<return value>	Class	The class of the associated property editor. May be null.

Returns the class of the associated property editor. The Layout Designer uses this to instantiate a property editor for the UI element. The default implementation returns a null value, indicating no associated property editor exists.

getRejectReason

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
<return value>	String	The reason text for a false value returned by the last `canAcceptChild` or `canAcceptParent` call. May be null.

When a **`canAcceptChild`** or **`canAcceptParent`** method call returns false, this method can be called to return a human-readable reason for the rejection. This value is reset upon each such method call and will be null if the method returned true.

getRoot

org.carewebframework.shell.layout.UIElementBase

Returns the UI element at the root of the component tree.

getSerializableChildren

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
<return value>	Iterable <UIElementBase>	List of children that can be serialized.

Returns a list of children that may be serialized. By default, this returns the same list as **`getChildren`**. Subclasses that need to modify this default behavior can override this method.

hasAncestor

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
element	UIElementBase	A UI element.
<return value>	boolean	True if the specified UI element is an ancestor of this element.

Use to determine if a UI element is an ancestor of this one.

hasChild

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
element	UIElementBase	A UI element.
<return value>	boolean	True if the specified UI element is an immediate child of this element.

Use to determine if a UI element is an immediate child of this one.

indexOfChild

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
child	UIElementBase	The child UI element whose index is sought.
<return value>	int	The index of the child in this UI element's list of children. If this UI element has no children, or if the specified child is not in the list, a value of -1 is returned.

Returns the index of the specified child in this UI element's list of children.

isActivated

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
<return value>	boolean	The activation state of this UI element.

Returns the activation state of this UI element.

isContainer

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
<return value>	boolean	True if this UI element can contain other UI elements.

Returns true if this UI element can act as a container for other UI elements.

isDesignMode

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
<return value>	boolean	True if this UI element is configured for design mode.

The Layout Designer sets the **`DesignMode`** property to true for every UI element in the current layout when it enters design mode and sets it to false when exiting design mode. This allows a UI element to make any necessary changes to its state to accommodate this special mode. This is the getter method for the **`DesignMode`** property.

isEnabled

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
<return value>	boolean	The enabled state for this UI element.

Returns the enabled state for this UI element.

isLocked

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
<return value>	boolean	The locked state for this UI element.

Returns the locked state for this UI element. If a UI element is locked, the Layout Designer will restrict the user's ability to manipulate the element in design mode.

isVisible

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
<return value>	boolean	The visibility state of the UI element.

Returns the visibility state of the UI component.

raise [S]

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
text	String	Text to be associated with the exception.
exception (optional)	Throwable	The cause of the exception.

Throws an exception of type **`UIException`** with the specified message text.

remove

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
destroy	boolean	If true, this element is explicitly destroyed after being removed.

Removes this UI element from its parent and optionally destroys it.

removeChild

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
child	UIElementBase	The child UI element that is about to be removed.
destroy	boolean	If true, the child is explicitly destroyed after being removed.

Removes the specified UI element as a child of this UI element. If **`destroy`** is true, the child is also explicitly destroyed (i.e., its **`destroy`** method is called) after it is removed.

removeChildren

org.carewebframework.shell.layout.UIElementBase

Removes (and destroys) all of this UI element's children.

setColor

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
value	String	The color as specified in HTML format.

Sets the color value.

setDefinition

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
clazz	Class <? extends UIElementBase>	Class whose associated definition is sought.

Sets the plugin definition to be associated with this UI element based on the specified class. Typically, the class would be the same as that of the element itself, but in certain cases (as in the **`UIElementProxy`** class) it is not. This method uses the plugin registry to retrieve the plugin definition that is associated with the specified class. This method raises a run time exception if the plugin definition has already been set.

setDefinition

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
definition	PluginDefinition	The plugin definition to be associated with this UI element.

Sets the plugin definition to be associated with this UI element. This method raises a run time exception if the plugin definition has already been set.

setDesignMode

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
designMode	boolean	The design mode state.

The Layout Designer sets the **`DesignMode`** property to true for every UI element in the current layout when it enters design mode and sets it to false when exiting design mode. This allows a UI element to make any necessary changes to its state to accommodate this special mode. This is the setter method for the **`DesignMode`** property.

setEnabled

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
enabled	boolean	The enabled state.

Sets the enabled state of the UI element. This base implementation only sets the internal flag and notifies the parent of the state change. Each subclass is responsible for overriding this method to properly reflect the enabled state in their wrapped UI components.

setHint

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
value	String	The hint text.

Sets the hint text.

setIndex

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
index	int	The new index for this UI element.

Sets the position of this UI element relative to its siblings.

setInnerComponent

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
value	Object	The wrapped UI component that will serve as parent to any added children.

Sets the wrapped UI component that will serve as a parent to any added children. If the inner and outer components are the same, only one of them needs to be set.

setLocked

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
locked	boolean	The new locked state.

Sets the locked state for this UI element. If a UI element is locked, the Layout Designer will restrict the user's ability to manipulate the element in design mode.

setOuterComponent

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
value	Object	The wrapped UI component that will be attached to the inner UI component of a parent.

Sets the wrapped UI component that will be attached to the inner component of the parent UI element. If the inner and outer components are the same, only one of them needs to be set.

setParent

org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
parent	UIElementBase	The new parent for this UI element.

Sets the parent for this UI element, subject to the parent/child constraints applicable to each.

setRejectReason org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
rejectReason	String	The reject reason message text.

Sets the text of the reason why a parent or child candidate was rejected.

setVisible org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
visible	boolean	The new visibility state.

Sets the visibility state of the UI element.

unbind org.carewebframework.shell.layout.UIElementBase

Override to unbind wrapped components from the UI.

updateParentState org.carewebframework.shell.layout.UIElementBase

Calls the **`updateState`** method on the parent UI element if one exists.

updateState org.carewebframework.shell.layout.UIElementBase

Update a UI element based on the state of its children. This has no effect on non-container elements. For elements that can act as containers, the default behavior is as follows:

- If all children are disabled, the parent is also disabled.
- If all children are hidden or there are no children and design mode is not active, the parent is also hidden.

updateVisibility org.carewebframework.shell.layout.UIElementBase

Calls the parameterized **`updateVisibility`** method with the current settings.

updateVisibility org.carewebframework.shell.layout.UIElementBase

Parameter	Datatype	Description
visible	boolean	The current visibility state.
activated	boolean	The current activation state.

Invoked when visibility or activation state changes. Override to set the visibility of wrapped components.

UIElementZKBase Class

This is the abstract base class for all UI elements that wrap ZK components. It is a subclass of the **`UIElementBase`** class and overrides several of its methods to provide generic handling for wrapped ZK components. For many subclasses of **`UIElementZKBase`**, these generic implementations will be sufficient. For others, some methods may need to be further overridden to provide the desired behaviors. **`UIElementZKBase`** also introduces several new methods specific to manipulating ZK components.

afterMoveTo

org.carewebframework.shell.layout.UIElementZKBase

Parameter	Datatype	Description
index	int	The new index.

Provides a default behavior for resequencing this element relative to its siblings. The implementation logic assumes that there is one wrapped ZK component and it is that component whose position needs to be changed.

applyColor

org.carewebframework.shell.layout.UIElementZKBase

Parameter	Datatype	Description
component	Object	The ZK component whose color is to be set.

Applies the current setting of the **`Color`** property to the background color of the specified ZK component. If the ZK component implements a **`setCustomColor`** JavaScript method, that method will be called to set the color. If the specified component is not a subclass of **`HtmlBasedComponent`**, the request is ignored.

applyHint

org.carewebframework.shell.layout.UIElementZKBase

Parameter	Datatype	Description
component	Object	The ZK component whose hint text is to be set.

Sets the tooltip text on the ZK component using the current value of the hint text. If the specified component is not a subclass of **`HtmlBasedComponent`**, the request is ignored.

associateComponent

org.carewebframework.shell.layout.UIElementZKBase

Parameter	Datatype	Description
component	Component	The ZK component to be associated with this UI element.

Associates the specified ZK component with this UI element, so that a call to **`getAssociatedUIElement`** on that component will return this UI element. This is useful for operations that require knowledge of the UI element wrapper given a ZK component.

bind

org.carewebframework.shell.layout.UIElementZKBase

Implements the default binding logic for ZK-based UI elements by appending the outer ZK component of this UI element to the inner ZK component of its parent. Override to accommodate more complex binding behavior.

createFromTemplate

org.carewebframework.shell.layout.UIElementZKBase

Parameter	Datatype	Description
template (optional)	**String**	**URL of the zul page that will serve as a template. If the URL is not specified, the template name is obtained from `getTemplateUrl`.**
parent (optional)	**Component**	**The ZK component that will become the parent. It may be null (the default).**
controller (optional)	**Object**	**If specified and not null, events and instance variables are auto-wired to this controller.**
<return value>	**Component**	**The top level ZK component produced by the template.**

Creates ZK component(s) from a template (a zul page). Performs auto-wiring of instance variables and events to a controller if one is specified. If a template URL is not specified or is null, it will be obtained by a call to **`getTemplateUrl`**.

editProperties

org.carewebframework.shell.layout.UIElementZKBase

Overrides the default implementation of the base class by invoking the generic property editor for this UI element.

fullSize

org.carewebframework.shell.layout.UIElementZKBase

Parameter	Datatype	Description
component	**HtmlBasedComponent**	**The ZK component whose dimensions are to be modified.**

This is a convenience method that sets the height and width of a ZK component to 100%.

getAssociatedUIElement [S]

org.carewebframework.shell.layout.UIElementZKBase

Parameter	Datatype	Description
component	**Component**	**The ZK component whose associated UI element is sought.**
<return value>	**UIElementBase**	**The UI element associated with the specified ZK component.**

Returns the UI element that is associated with the specified ZK component, or null if there is no such association.

getEnableDesignModeMask

org.carewebframework.shell.layout.UIElementZKBase

Parameter	Datatype	Description
<return value>	**boolean**	**True if the design mode mask is enabled.**

Returns true if this UI element is to display a special mask during design mode. When enabled, activating design mode causes a translucent mask to appear over the component that prevents direct user interaction.

getTemplateUrl

org.carewebframework.shell.layout.UIElementZKBase

Parameter	Datatype	Description
<return value>	**String**	**The default template URL.**

Returns the default template URL to be used by **`createFromTemplate`** when no URL is specified. In this implementation, the template URL is derived from the UI element's class name. For example, if the class is **`org.acme.xxx.MyClass`**, the default

template URL is computed to be `~./org/acme/xxx/MyClazz.zul`. Override this method to provide an alternative default URL.

hasVisibleElements

org.carewebframework.shell.layout.UIElementZKBase

Parameter	Datatype	Description
component	Component	Root of the component subtree to be scanned.
<return value>	boolean	True if any associated UI element in the component subtree is visible.

Returns true if any associated UI element in the component subtree is visible.

moveChild

org.carewebframework.shell.layout.UIElementZKBase

Parameter	Datatype	Description
child	Component	The child ZK component that will be moved.
index	int	The index (relative to its siblings) where the child component will be moved.

Moves a child ZK component to a new position under its current parent.

setDesignContextMenu

org.carewebframework.shell.layout.UIElementZKBase

Parameter	Datatype	Description
contextMenu	Menupopup	The Layout Designer context menu if design mode is activated, null if deactivated.

The Layout Designer uses this method to apply its context menu to individual UI elements when design mode is activated and remove it when inactivated. The default implementation calls the overloaded version of this method, passing the outer ZK component as the target.

setDesignContextMenu

org.carewebframework.shell.layout.UIElementZKBase

Parameter	Datatype	Description
component	Component	Component associated with the context menu.
contextMenu	Menupopup	The Layout Designer context menu if design mode is activated, null if deactivated.

The Layout Designer uses this method to apply its context menu to a target ZK component.

setDesignMode

org.carewebframework.shell.layout.UIElementZKBase

Parameter	Datatype	Description
designMode	boolean	The design mode state.

The Layout Designer sets the `DesignMode` property to true for every UI element in the current layout when it enters design mode and sets it to false when exiting design mode. This augments the base implementation by applying or removing the Layout Designer's context menu.

setEnableDesignModeMask org.carewebframework.shell.layout.UIElementZKBase

Parameter	Datatype	Description
value	boolean	True enables the design mode mask.

Sets the value for enabling the design mode mask. See the **`getEnableDesignModeMask`** method for more information.

setInnerComponent org.carewebframework.shell.layout.UIElementZKBase

Parameter	Datatype	Description
value	Object	The wrapped UI component that will serve as parent to any added children.

Sets the wrapped UI component that will serve as a parent to any added children. This augments the base implementation by associating the component with this UI element via the **`associateComponent`** method.

setOuterComponent org.carewebframework.shell.layout.UIElementZKBase

Parameter	Datatype	Description
value	Object	The wrapped UI component that will be attached to the inner UI component of a parent.

Sets the wrapped UI component that will be attached to the inner component of the parent UI element. This augments the base implementation by associating the component with this UI element via the **`associateComponent`** method.

swapChildren org.carewebframework.shell.layout.UIElementZKBase

Parameter	Datatype	Description
child1	Component	The first ZK component to be swapped.
child2	Component	The second ZK component to be swapped.

Swaps the position of the two child ZK components.

unbind org.carewebframework.shell.layout.UIElementZKBase

Detaches wrapped ZK components. The default implementation detaches the outer component only.

updateVisibility org.carewebframework.shell.layout.UIElementZKBase

Parameter	Datatype	Description
visible	boolean	The visibility state.
activated	boolean	The activation state.

Provides a default implementation where the outer ZK component is made visible if both its visibility and its activation states are both true.

Other UI Element Classes

From the aforementioned base classes are derived a number of UI elements that are packaged with the CWF. Some of these UI elements have special uses (like **`UIElementPlugin`** and **`UIElementProxy`**), but most correspond to visual elements

that can be directly used and referenced within a layout. Some UI elements can act as containers for other UI elements. For those, many have complementary classes, one for the base view (e.g., **`UIElementSplitterView`**) and one for each of the corresponding containers (e.g., **`UIElementSplitterPane`**).

An inventory of available UI elements follows, including some basic information about each.

UIElementButton

class	org.carewebframework.shell.layout.UIElementButton	
tag	button	
parent	any	
children	none	
properties	action	An action to be invoked when the button is clicked.
	hint	The tool tip text to display upon hover.
	icon	The URL of an icon to display on the button.
	label	The text label for the button.

Wraps the ZK button component. See the section on actions for more information on the **`action`** property.

UIElementDesktop

class	org.carewebframework.shell.layout.UIElementDesktop	
tag	_desktop	
parent	none	
children	any	
properties	icon	The URL of an icon to display in the title bar.
	menubar	A pseudo-property for managing custom menu items.
	title	The title text appearing in the application title bar.
	toolbar	A pseudo-property for managing the tool bar contents.

This is the top level UI element for any layout. It is an implicitly created UI element (as evidenced by its tag beginning with an underscore). The application id is used when referencing application-specific property settings.

UIElementFrame

class	org.carewebframework.shell.layout.UIElementFrame	
tag	frame	
parent	any	
children	none	
properties	url	A URL to a web resource.

This UI element wraps a ZK **`Iframe`** (if the URL specifies the http or https protocol) or **`Include`** component. It may be used to display arbitrary web content (internal or external) within the application.

UIElementLayout

class	org.carewebframework.shell.layout.UIElementLayout	
tag	layout	
parent	any	
children	any	
properties	shared	If true, this is a shared (public) layout. If false, only the owner may access the layout.
	layoutName	A unique name of the layout.
	linked	If true, the layout is linked. If false, it is embedded.

This UI element identifies a layout. A layout is a named snapshot of a UI layout that may be included within another layout. In linked mode, only a reference to the layout is stored. Any changes to the original layout are reflected in the referencing layout. In embedded mode, a copy of the entire layout (including all children) is stored. Any changes to the original layout will not be reflected in the referencing layout.

UIElementLink

class	org.carewebframework.shell.layout.UIElementLink	
tag	link	
parent	any	
children	none	
properties	action	An action to be invoked when the button is clicked.
	color	The color to be applied to the link.
	hint	The tool tip text to display upon hover.
	icon	The URL of an icon to be displayed with the link.
	label	The text label representing the link.

This wraps ZK anchor component, representing an actionable link. See the section on actions for more information on the **`action`** property.

UIElementMenubar

class	org.carewebframework.shell.layout.UIElementMenubar	
tag	_menubar or menubar	
parent	any	
children	UIElementMenuItem	
properties	menuitem	A pseudo-property for managing menu items.

This wraps ZK's **`Menubar`** component. An instance of this UI element is implicitly created and attached to the desktop element, but additional instances may be created elsewhere in the UI if desired (hence, the two tags). The **`menuitem`** property is a pseudo-property with an associated custom property editor that knows how to manage menu items.

UIElementMenuItem

class	org.carewebframework.shell.layout.UIElementMenuItem	
tag	menuitem	
parent	UIElementMenubar or UIElementMenuItem	
children	UIElementMenuItem	
properties	action	An action to be invoked when the item is clicked.
	color	The color of label text.
	hint	The tool tip text to display upon hover.
	label	The label text associated with menu item.

This wraps ZK's **`Menuitem`** component. See the section on actions for more information on the **`action`** property.

UIElementPlugin

class	org.carewebframework.shell.layout.UIElementPlugin
tag	plugin-dependent
parent	any
children	none
properties	plugin-dependent

A subclass of **`UIElementZKBase`**, this is the UI element that underlies simple CWF UI plugins. Its wrapped ZK component is of the class **`PluginContainer`**, which acts as a container for a CWF plugin's principal zul page. Most of the method overrides simply delegate the corresponding function to the container. For example, property resolution is delegated to the container, which maintains a list of properties and their associated beans as registered by its hosted plugin. So the UI element delegates property resolution to the container, which, in turn, delegates it to the bean registered by the hosted plugin. A more detailed discussion of the **`PluginContainer`** class may be found in the section on plugin development.

UIElementProxy

class	org.carewebframework.shell.layout.UIElementProxy
tag	same as the proxied UI element
parent	same as the proxied UI element
children	same as the proxied UI element
properties	same as the proxied UI element

A subclass of **`UIElementBase`**, this is a special UI element used by the Layout Designer as a proxy for another UI element. This allows the Layout Designer to 1) buffer changes to an existing UI element, and 2) to defer creation of a new UI element while allowing its proxy to be configured. At the user's direction, the Layout Designer may then apply changes made to the proxy to the underlying UI element, or discard the changes altogether.

UIElementSplitterPane

class	org.carewebframework.shell.layout.UIElementSplitterPane
tag	splitterpane
parent	UIElementSplitterView

children	any	
properties	caption	The text that will appear at the top of the splitter pane.
	color	The background color of the pane.
	relative	If true, the pane's size will change proportionately as its parent's size changes. If false, the pane's size is fixed.
	size	The size (width or height, depending on the splitter view's orientation) of the pane. For relative dimensions, this is a percentage value; for fixed it is measured in pixels.

This is the container element for the splitter view UI element.

UIElementSplitterView

class	org.carewebframework.shell.layout.UIElementSplitterView	
tag	splitterview	
parent	any	
children	UIElementSplitterPane	
properties	color	The background color of the splitter view.
	orientation	The orientation, which may be "horizontal" or "vertical".
	panes	A pseudo-property for managing splitter panes.

This UI element supports a view of multiple panes, separated by adjustable splitter bars, in a horizontal or vertical orientation. Each pane may serve as a container to another UI element.

UIElementStepPane

class	org.carewebframework.shell.layout.UIElementStepPane	
tag	steppane	
parent	UIElementStepView	
children	any	
properties	color	The background color of the task button.
	hint	The text that appears when the cursor hovers over the task button.
	icon	The URL of the icon that appears on the task button
	label	The text that appears on the task button.

This is the container element for the step view UI element.

UIElementStepView

class	org.carewebframework.shell.layout.UIElementStepView	
tag	stepview	
parent	any	
children	UIElementStepPane	
properties	caption	The caption text.
	color	The background color of the step view.
	noHome	If true, hides the home button. The pane associated with the home button is displayed as a normal step

		pane.
	noNavigation	If true, the navigational elements are hidden.
	steps	A pseudo-property for managing individual step panes.

This UI element supports a step-wise view of multiple panes. Each pane may serve as a container to another UI element and only one pane may be active at a time. A step selector is provided at the top. This is a sequence of buttons, connected visually by unidirectional arrows. The user clicks a button to activate the associated pane.

UIElementTabMenu

class	org.carewebframework.shell.layout.UIElementTabMenu	
tag	tabmenu	
parent	UIElementTabPane	
children	UIElementTabMenuPane	
properties	menuitems	A pseudo-property for managing menu items.

This UI element may be placed on a tab pane, providing a drop down menu on the associated tab for selecting menu panes.

UIElementTabMenuPane

class	org.carewebframework.shell.layout.UIElementTabMenuPane	
tag	tabmenupane	
parent	UIElementTabMenu or UIElementTabMenuPane	
children	any	
properties	color	The background color of the menu pane.
	hint	The text that appears when the cursor hovers over the menu item.
	label	The text that appears on the menu item.

This is the child element for the tab menu UI element. It consists of a menu item and an associated pane. The pane is activated when the menu item is clicked.

UIElementTabPane

class	org.carewebframework.shell.layout.UIElementTabPane	
tag	tabpane	
parent	UIElementTabView	
children	any	
properties	color	The background color of the tab.
	hint	The text that appears when the cursor hovers over the tab.
	label	The text that appears on the tab.

This is the container element for the tab view UI element.

UIElementTabView

class	org.carewebframework.shell.layout.UIElementTabView	
tag	tabview	
parent	any	
children	UIElementTabPane	
properties	color	The background color of the tab view.
	orientation	The orientation, which may be "horizontal" or "vertical".
	tabs	A pseudo-property for managing individual tabs.

This UI element supports a view of multiple tabs in a horizontal (tabs along the top) or vertical (tabs along the left) orientation. Each tab may serve as a container to another UI element and only one tab may be active at a time.

UIElementToolbar

class	org.carewebframework.shell.layout.UIElementToolbar	
tag	_toolbar or toolbar	
parent	any	
children	any	
properties	children	A pseudo-property for managing tool bar content.

This wraps ZK's toolbar component. An instance of this UI element is implicitly created and attached to the desktop element, but additional instances may be created elsewhere in the UI if desired (hence, the two tags). The **`children`** property is a pseudo-property with an associated custom property editor for managing child UI elements placed on the toolbar. While the toolbar element does not constrain which UI elements may be placed upon it, it is not suitable for hosting all UI elements and should generally be limited to elements such as buttons and hyperlinks.

UIElementTreePane

class	org.carewebframework.shell.layout.UIElementTreePane	
tag	treepane	
parent	UIElementTreeView or UIElementTreePane	
children	any	
properties	color	The background color of the pane.
	hint	The text that appears when the cursor hovers over the tree node.
	label	The label text for tree node.

This is the container element for the tree view UI element.

UIElementTreeView

class	org.carewebframework.shell.layout.UIElementTreeView	
tag	treeview	
parent	any	
children	UIElementTreePane	
properties	caption	The text that appears at the top of the tree view selector.
	color	The background color of the tree view.
	open	Controls whether the tree view selector pane is open or collapsed.
	panes	A pseudo-property for managing individual panes.

This UI element supports a tree-based view of multiple panes. Each pane may serve as a container to another UI element and only one pane may be active at a time. A tree view selector on the left panel allows the user to select which associated pane is to be activated.

Layouts

The Layout Manager enables the creation, manipulation, and visual rendering of UI layouts. Because all UI elements descend from a common base class and are self-describing, the Layout Manager understands each element's capabilities and how to instantiate, serialize and manipulate it. Layouts may be created programmatically, from serialized XML-based snapshots, interactively via a special design mode feature, or a combination of these approaches.

The **`CareWebShell`** class provides the base canvas upon which all layouts are built. The **`CareWebShellEx`** class, a subclass of **`CareWebShell`**, provides a convenient default initial layout consisting of a top-level tab view with a tree view on each tab and methods for easily adding plugins to the layout. If this default layout suits your needs, the **`CareWebShellEx`** class may be a better choice. For more flexible layout capabilities, use **`CareWebShell`** instead. Both classes are described here.

Both **`CareWebShell`** and **`CareWebShellEx`** are designed to be directly imbedded into a zul page via their ZK tag references, **`cwfShell`** and **`cwfShellEx`**, respectively. This is illustrated by the following examples:

```
<zk>
   <cwfShell id="careWebShell" height="100%" width="100%"
      layout="/layout.xml" />
</zk>
```

or

```
<zk>
   <cwfShellEx id="careWebShell" height="100%" width="100%">
      <attribute name="onCreate">
         <!-- initialization code here -->
      </attribute
   </cwfShellEx>
</zk>
```

The **`onCreate`** attribute provides a convenient place to initialize your layout, especially if you are building it programmatically. The **`layout`** property allows you to reference an XML-based snapshot from which to construct a layout.

CareWebShell Class

The **`CareWebShell`** class declares the following methods:

addMenu

org.carewebframework.shell.CareWebShell

Parameter	Datatype	Description
path	**String**	**The backslash-delimited path for the menu item.**
action	**String**	**An action to perform when the menu item is clicked.**
<return value>	**Menu**	**The newly created menu.**

Adds a menu to the shared desktop menu. See the section on actions for more information on the **`action`** property.

addToolbarComponent

org.carewebframework.shell.CareWebShell

Parameter	Datatype	Description
component	**Component**	**The ZK component to add.**

Adds a ZK component to the shared desktop toolbar.

buildUI

org.carewebframework.shell.CareWebShell

Parameter	Datatype	Description
layout	**UILayout**	**The layout to be built.**

Removes any existing layout and renders a new UI based on the specified layout. The **`UILayout`** class is the in-memory representation of an xml-based layout. It is described in detail later in this section.

clearMessages

org.carewebframework.shell.CareWebShell

Parameter	Datatype	Description
tag (optional)	**String**	**If specified, only messages with this tag will be cleared. Otherwise, all messages are cleared.**

Clears slide-down messages.

getActivatedPlugin

org.carewebframework.shell.CareWebShell

Parameter	Datatype	Description
id	**String**	**The unique id of the plugin.**
<return value>	**PluginContainer**	**The activated plugin matching the specified id.**

Searches for an activated plugin with the specified id, returning its container if found or null if not. If more than one activated plugin matches the specified id, only the first one found is returned.

getActivatedPlugins

org.carewebframework.shell.CareWebShell

Parameter	Datatype	Description
<return value>	**Iterable <PluginContainer>**	**An iterable of all activated plugins. Never null.**

Returns an iterable of all activated plugins. The result may contain no elements, but will never be null.

getActivatedPlugins

org.carewebframework.shell.CareWebShell

Parameter	Datatype	Description
list	Collection <PluginContainer>	A collection that will receive the results. If null, a new collection will be created.
<return value>	Collection <PluginContainer>	The collection that received the list of activated plugins.

Populates a collection (or creates a new one) with a list of all activated plugins.

getClientInformation

org.carewebframework.shell.CareWebShell

Parameter	Datatype	Description
<return value>	ClientInfoEvent	Information about the browser client.

Returns information about the browser client in ZK's `ClientInfoEvent` object. This class provides information about the browser's dimensions, color depth, and time zone. See ZK's documentation for additional detail.

getLayout

org.carewebframework.shell.CareWebShell

Parameter	Datatype	Description
<return value>	String	Name of the layout to be loaded.

Returns the name of the layout that will be loaded upon initialization.

getLoadedPlugin

org.carewebframework.shell.CareWebShell

Parameter	Datatype	Description
id	String	The unique id of the plugin.
forceInit (optional)	boolean	If true and the plugin has not been initialized, it will be initialized at this time. If not specified, defaults to false.
<return value>	PluginContainer	The loaded plugin matching the specified id, or null if not found.

Searches for a loaded plugin with the specified id, returning its container if found or null if not. If more than one loaded plugin matches the specified id, only the first one found is returned. If `forceInit` is specified and is true, the plugin will be initialized if not already so.

getLoadedPluginDefinitions

org.carewebframework.shell.CareWebShell

Parameter	Datatype	Description
<return value>	Iterable <PluginDefinition>	An iterable of plugin definitions for all loaded plugins. Never null.

Returns an iterable of plugin definitions for all loaded plugins. If more than one loaded plugin exists for a given definition, the definition will occur only once in the returned results.

getLoadedPlugins

org.carewebframework.shell.CareWebShell

Parameter	Datatype	Description
<return value>	Iterable <PluginContainer>	An iterable of all loaded plugins. Never null.

Returns an iterable of all loaded plugins. The result may contain no elements, but will never be null.

getPropertyGroups

org.carewebframework.shell.CareWebShell

Parameter	Datatype	Description
<return value>	List <String>	A list of property groups bound to loaded plugins.

Returns a list of all property groups bound to loaded plugins. No property group will appear more than once in the list.

getUIDesktop

org.carewebframework.shell.CareWebShell

Parameter	Datatype	Description
<return value>	UIElementDesktop	A reference to the top-level desktop.

Returns a reference to the top-level desktop of the layout.

getUILayout

org.carewebframework.shell.CareWebShell

Parameter	Datatype	Description
<return value>	UILayout	The last UI layout that was loaded. May be null.

Returns a reference to the last UI layout that was loaded. If no UI layout has been loaded, this will return null.

isAutoStart

org.carewebframework.shell.CareWebShell

Parameter	Datatype	Description
<return value>	boolean	The `autostart` setting. Defaults to false.

Returns the **`autostart`** setting. If true, the **`start`** method will be invoked immediately after loading a new layout. If false, the **`start`** method must be invoked manually.

loadLayoutByAppId

org.carewebframework.shell.CareWebShell

Parameter	Datatype	Description
appId (optional)	String	The id of the application whose associated layout is to be loaded. If not specified, the `appId` is taken from the `appId` query parameter of the application's URL or, if not specified there, from the `CAREWEB.APPID.DEFAULT` domain property.

Renders the UI based on the layout associated with the specified **`appId`**. If an **`appId`** is not specified, the application looks for a query parameter by the same name for this value or, failing that, for a domain property named **`CAREWEB.APPID.DEFAULT`**.

loadLayoutFromResource

org.carewebframework.shell.CareWebShell

Parameter	Datatype	Description
resource	String	A resource specifier (see below).

Loads the XML-based layout from the specified resource and renders the UI based on the layout. The resource type is determined by the presence of one of the following prefix values:

- app:xxx – References the layout associated with the application id "xxx".
- shared:xxx – References the shared layout named "xxx".

- private:xxx – References the private layout named "xxx".
- Otherwise, the resource specifier is assumed to be a URL.

lock

org.carewebframework.shell.CareWebShell

Locks the desktop. Locking the desktop hides the desktop contents with a privacy screen that can only be removed by entering the current user's password.

logout

org.carewebframework.shell.CareWebShell

Logs the user out after presenting a confirmation prompt (which may be suppressed by user preference).

registerStyleSheet

org.carewebframework.shell.CareWebShell

Parameter	Datatype	Description
url	**String**	**URL of an external style sheet.**

Registers an external style sheet by adding a reference to the current ZK page.

reset

org.carewebframework.shell.CareWebShell

Resets the UI desktop to its baseline state, removing the loaded layout.

setAutoStart

org.carewebframework.shell.CareWebShell

Parameter	Datatype	Description
autoStart	**boolean**	**The new `autostart` setting.**

Sets the **`autostart`** setting. If true, the **`start`** method will be invoked immediately after loading a new layout. If false, the **`start`** method must be invoked manually.

setLayout

org.carewebframework.shell.CareWebShell

Parameter	Datatype	Description
defaultLayoutName	**String**	**The name of the layout to be loaded at initialization.**

Sets the name of the layout to be loaded at initialization. If initialization has already occurred, the named layout is loaded immediately.

showMessage

org.carewebframework.shell.CareWebShell

Parameter	Datatype	Description
message	**String**	**The message text.**
caption (optional)	**String**	**The caption text for the message.**
color (optional)	**String**	**The background color for the message.**
duration (optional)	**Integer**	**The duration of message in milliseconds. Defaults to 8000.**
tag (optional)	**String**	**The tag to associate with the message (for conditional clearing of messages by tag name).**

Displays a slide-down message. These occur as an animated slide-down panel from the top center of the view port.

start org.carewebframework.shell.CareWebShell

The primary purpose of this method is to execute any registered startup routines. Any object of a class that implements the **`ICareWebStartup`** interface and is registered with the CWF can serve as a startup routine. This interface declares a single method, **`execute`**, that should return true if the startup routine is to be executed only once or false if it is to be executed every time the **`start`** method is invoked.

CareWebShellEx Class

The **`CareWebShellEx`** class extends **`CareWebShell`**. It provides a default layout consisting of a tab view with each tab containing a tree view. It has several methods that simplify the work of adding plugins to this default layout. Many of these methods require a **`path`** parameter that specifies where the plugin is to be added to the layout. This **`path`** parameter consists of one or more labels, delimited by backslashes. The first label determines the tab on which the plugin will reside. If no tab with a matching label exists, one will be created. Subsequent labels determine the location of the plugin in the tree view on the associated tab. If only one label is specified, the plugin is positioned directly on the tab without a tree view. A tab can either host a plugin or a tree view, but not both.

A special case exists where if the **`path`** parameter has the value "@toolbar", the plugin is added to the shared toolbar.

The **`CareWebShellEx`** class has the following methods:

getPathResolver org.carewebframework.shell.CareWebShellEx

Parameter	Datatype	Description
<return value>	PathResolver	The path resolver.

Returns the path resolver to use to determine where a plugin should be placed based on its path. While the first part of a path always refers to a tab, the path resolver determines how the rest of the path maps to the layout. The default path resolver maps into a tree view component. You may create an alternative path resolver that uses a different mapping.

The **`PathResolver`** constructor takes two arguments. The first is the class of the parent UI element and the second is the class of its child elements. The child element class must expose a **`label`** property. The default path resolver uses **`UIElementTreeView`** and **`UIElementTreePane`**, respectively, for the two constructor arguments.

register org.carewebframework.shell.CareWebShellEx

Parameter	Datatype	Description
path	String	The path that determines where the plugin will be placed.
def	PluginDefinition	The plugin's definition.
propertySource (optional)	IPropertyProvider	An optional source for retrieving property values to assign to the plugin.
<return value>	UIElementBase	The UI element that hosts the plugin.

Creates an instance of a plugin given its definition and adds it to the UI in the location specified by the **`path`** parameter. If a **`propertySource`** is specified, the plugin's property values are initialized using that source.

register org.carewebframework.shell.CareWebShellEx

Parameter	Datatype	Description
path	String	The path that determines where the plugin will be placed.
url	String	The URL of the plugin's top-level zul page.
propertySource (optional)	IPropertyProvider	An optional source for retrieving property values to assign to the plugin.
<return value>	UIElementBase	The UI element that hosts the plugin.

Creates an instance of a plugin given the URL of its top-level zul page and adds it to the UI in the location specified by the **`path`** parameter. If a **`propertySource`** is specified, the plugin's property values are initialized using that source.

registerFromId org.carewebframework.shell.CareWebShellEx

Parameter	Datatype	Description
path	String	The path that determines where the plugin will be placed.
id	String	The plugin's unique id.
propertySource (optional)	IPropertyProvider	An optional source for retrieving property values to assign to the plugin.
<return value>	UIElementBase	The UI element that hosts the plugin.

Creates an instance of a plugin given its unique id and adds it to the UI in the location specified by the **`path`** parameter. If a **`propertySource`** is specified, the plugin's property values are initialized using that source.

registerLayout org.carewebframework.shell.CareWebShellEx

Parameter	Datatype	Description
path	String	The path that determines where the plugin will be placed.
url	String	The URL of the XML-based layout.

Creates a UI layout from the specified URL and builds a UI based on that layout at the location specified by the **`path`** parameter.

registerMenu org.carewebframework.shell.CareWebShellEx

Parameter	Datatype	Description
menuPath	String	A backslash-delimited path that indicates where the menu item will reside in the shared menu bar.
action	String	The action to be invoked when the menu item is clicked.
<return value>	UIElementMenuItem	The UI element that wraps the newly created menu item.

Creates a new menu item and positions it in the shared menu bar in the location specified by the **`menuPath`** parameter.

setPathResolver org.carewebframework.shell.CareWebShellEx

Parameter	Datatype	Description
pathResolver	PathResolver	The path resolver.

Replaces the default path resolver. See the **`getPathResolver`** method for more information.

UILayout Class

We continue our discussion of layouts with a description of XML-based layouts and their in-memory representation as defined by the **`UILayout`** class. A UI layout may be persisted to and loaded from an XML representation, which may come from an external file, a stored property, or some other source. Given a layout, the Layout Manager may construct a UI interface based on that layout.

The following represents an example of a layout that might be used by the test harness:

```
<layout name="TESTHARNESS" title="CareWeb Test Harness" version="1.0">
    <_menubar />
    <_toolbar />
    <splitterview orientation="vertical">
        <splitterpane size="47" relative="false">
            <splitterview orientation="horizontal">
                <splitterpane size="50" relative="true">
                    <patientHeader />
                </splitterpane>
                <splitterpane size="50" relative="true">
                    <userHeader />
                </splitterpane>
            </splitterview>
        </splitterpane>
        <splitterpane size="90" relative="true">
            <tabview orientation="horizontal">
                <tabpane label="Test Harness">
                    <treeview caption="Detected Plugins:">
                    </treeview>
                </tabpane>
            </tabview>
        </splitterpane>
        <splitterpane size="17" relative="false">
            <statusPanel />
        </splitterpane>
    </splitterview>
</layout>
```

It should be fairly easy to see how this layout definition would translate to a UI. Other than the root tag, **`layout`**, each tag name corresponds to the id of a plugin or UI element. The attributes of each tag represent property values and the nesting levels represent parent-child relationships. The Layout Manager can take this information and its knowledge of registered plugins and UI elements and render a complete UI. Before the Layout Manager can consume an XML-based layout, that layout must be read into an instance of the **`UILayout`** class, which has the following methods of interest:

clear

org.carewebframework.shell.layout.UILayout

Removes any existing layout information.

deserialize

org.carewebframework.shell.layout.UILayout

Parameter	Datatype	Description
parent	**UIElementBase**	**The UI element that is to serve as the parent for the deserialized layout. May be null.**
<return value>	**UIElementBase**	**The top-level UI element that was created during deserialization.**

Deserializes the layout. Using information from the layout, this process creates each UI element, initializes its property values, and establishes the necessary parent-child relationships.

getName

org.carewebframework.shell.layout.UILayout

Parameter	Datatype	Description
<return value>	String	The name of this layout. May be null.

Returns the name of the layout. In the XML, this is given by the **`name`** attribute of the root **`layout`** tag.

getVersion

org.carewebframework.shell.layout.UILayout

Parameter	Datatype	Description
<return value>	String	The version of this layout.

Returns the version of the layout. In the XML, this is given by the **`version`** attribute of the root **`layout`** tag. This information may be used in the future to maintain backward compatibility with older formats of serialized layouts.

isEmpty

org.carewebframework.shell.layout.UILayout

Parameter	Datatype	Description
<return value>	boolean	True if the layout has no content.

Returns true if the layout has no content.

loadByAppId

org.carewebframework.shell.layout.UILayout

Parameter	Datatype	Description
appId	String	The application id whose associated layout is to be loaded.
<return value>	UILayout	This layout instance (for chaining).

Loads the layout associated with the specified application id. These associations are stored in a domain property called **`CAREWEB.LAYOUT.ASSOCIATION`**. This property stores, by application id, the name of the layout that is associated with the application. Once the layout name is known, this method calls **`loadFromProperty`** to retrieve the layout.

loadFromProperty

org.carewebframework.shell.layout.UILayout

Parameter	Datatype	Description
layoutName	String	The name of the layout to be loaded.
shared	boolean	If true, this is a shared layout; otherwise, it is a private layout.
<return value>	UILayout	This layout instance (for chaining).

Loads the named layout from the domain property store. Layouts are loaded from the **`CAREWEB.LAYOUT.SHARED`** or **`CAREWEB.LAYOUT.PRIVATE`** domain property using the layout name as the instance name.

loadFromText

org.carewebframework.shell.layout.UILayout

Parameter	Datatype	Description
text	String	The text of the XML-based layout.
<return value>	UILayout	This layout instance (for chaining).

Parses the XML input and loads it into the layout.

loadFromUrl

org.carewebframework.shell.layout.UILayout

Parameter	Datatype	Description
url	String	The URL of the resource containing the layout XML.

Loads a layout from an XML resource.

saveToProperty

org.carewebframework.shell.layout.UILayout

Parameter	Datatype	Description
layoutName	String	The name of the layout to be saved.
shared	boolean	If true, save as shared layout; otherwise, save as private.
<return value>	boolean	True if operation succeeded.

Saves the named layout to the domain property store. Layouts are stored in the **`CAREWEB.LAYOUT.SHARED`** or **`CAREWEB.LAYOUT.PRIVATE`** domain property using the layout name as the instance name.

serialize [S]

org.carewebframework.shell.layout.UILayout

Parameter	Datatype	Description
parent	UIElementBase	Top level element to be serialized.
<return value>	UILayout	A `UILayout` instance representing the serialized UI element tree.

Creates a **`UILayout`** instance based on the UI element subtree rooted at the specified parent element.

setName

org.carewebframework.shell.layout.UILayout

Parameter	Datatype	Description
name	String	The name of this layout. May be null.

Sets the name of the layout. In the XML, this is given by the **`name`** attribute of the root **`layout`** tag.

setVersion

org.carewebframework.shell.layout.UILayout

Parameter	Datatype	Description
version	String	The version of this layout.

Sets the version of the layout. In the XML, this is given by the **`version`** attribute of the root **`layout`** tag. This information may be used in the future to maintain backward compatibility with older formats of serialized layouts.

toString org.carewebframework.shell.layout.UILayout

Parameter	Datatype	Description
<return value>	String	The XML representation of the layout.

Returns the XML representation of the layout.

LayoutUtil Class

The **`LayoutUtil`** class is a library of static methods for convenient access to the Layout Services component, which provides functionality for managing layout persistence. The default implementation of the Layout Services component utilizes the domain property store for persisting layouts. Alternative implementations that utilize a different data store are possible.

The **`LayoutUtil`** class has the following static methods:

cloneLayout [S] org.carewebframework.shell.layout.LayoutUtil

Parameter	Datatype	Description
oldName	String	The name of the layout to clone.
newName	String	The name of the cloned layout.
shared	boolean	Whether this is a shared or private layout.

Clones a layout.

deleteLayout [S] org.carewebframework.shell.layout.LayoutUtil

Parameter	Datatype	Description
name	String	The name of the layout to delete.
shared	boolean	Whether this is a shared or private layout.

Deletes a layout.

getLayout [S] org.carewebframework.shell.layout.LayoutUtil

Parameter	Datatype	Description
name	String	The name of the layout to retrieve.
shared	boolean	Whether this is a shared or private layout.
<return value>	String	The layout content.

Retrieves the content of the named layout.

getLayoutByAppId [S] org.carewebframework.shell.layout.LayoutUtil

Parameter	Datatype	Description
appId	String	The application id.
<return value>	String	The layout content.

Retrieves the content of the layout associated with the specified application identifier.

getLayouts [S] org.carewebframework.shell.layout.LayoutUtil

Parameter	Datatype	Description
shared	boolean	Whether this is a shared or private layout.
<return value>	List <String>	List of layout names.

Returns a list of known shared or private layouts.

getLayoutService [S] org.carewebframework.shell.layout.LayoutUtil

Parameter	Datatype	Description
<return value>	ILayoutService	A reference to the Layout Service.

Returns a reference to the Layout Service.

layoutExists [S] org.carewebframework.shell.layout.LayoutUtil

Parameter	Datatype	Description
name	String	Layout name.
shared	boolean	Whether this is a shared or private layout.
<return value>	boolean	True if the specified layout exists.

Determines the existence of the named layout.

renameLayout [S] org.carewebframework.shell.layout.LayoutUtil

Parameter	Datatype	Description
oldName	String	The original layout name.
newName	String	The new layout name.
shared	boolean	Whether this is a shared or private layout.

Renames a layout.

saveLayout [S] org.carewebframework.shell.layout.LayoutUtil

Parameter	Datatype	Description
name	**String**	**The layout name.**
content	**String**	**The layout content.**
shared	**boolean**	**Whether this is a shared or private layout.**

Saves a layout.

validateName [S] org.carewebframework.shell.layout.LayoutUtil

Parameter	Datatype	Description
name	**String**	**The layout name to validate.**
<return value>	**boolean**	**True if the name is valid.**

Validates a layout name. A layout name must not be null or empty and may contain any alphanumeric characters, spaces or underscores.

Writing UI Plugins

Writing a UI plugin component for the CWF is a relatively straightforward process thanks to the **`UIElementPlugin`** class, which takes care of much of the plumbing required by the framework environment. For purposes of this discussion, we will assume that Eclipse is being used for software development (the descriptions that follow further assume the Kepler distribution of Eclipse) and that Maven support has been installed. The Spring Tool Suite version of Eclipse is highly recommended as it comes preconfigured with all the necessary tools. Users of other IDE's will need to adjust accordingly.

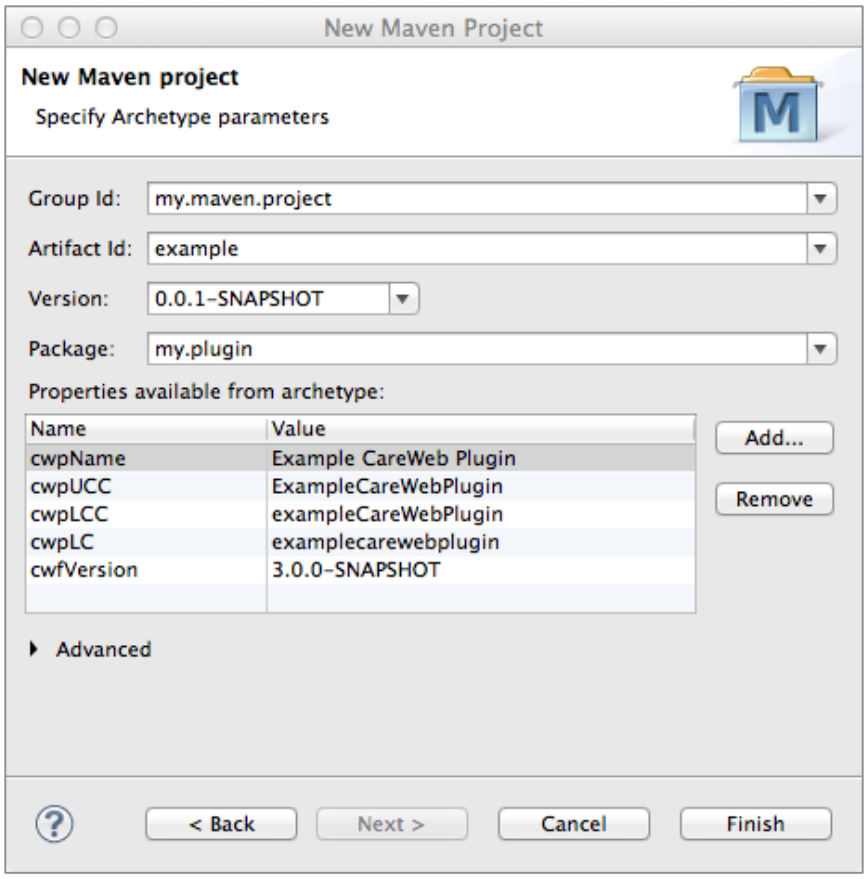
The CWF distribution contains two modules to aid in plugin development. The first is a Maven archetype for creating a skeleton plugin project and is a good starting point for creating a new plugin. The second is a test harness that can be used to test your component in a simulated deployment environment. We will describe each of these in more detail.

Creating a UI Plugin Project

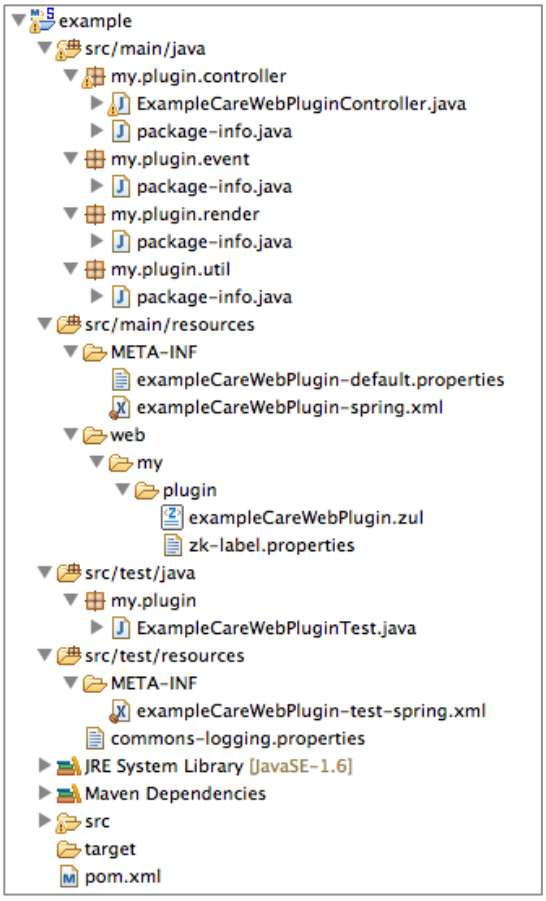
First, let us create a skeleton plugin project using the Maven archetype supplied with the CWF distribution. From the main Eclipse menu click **`File→New→Other…`** and select Maven Project from the dialog. From the New Maven Project dialog that appears, make certain that the "**`Create a simple project`**" check box is unchecked and click **`Next>`**. The next dialog allows you to select the Maven archetype to use to create the project. Make sure that **`All Catalogs`** is selected and enter "**`careweb`**" in the filter box. The CWF plugin archetype should appear in the list of matching entries. If it does not, try checking the "**`Include snapshot archetypes`**" check box. Select the entry with an artifact id of

`org.carewebframework.mvn.archetype.plugin`

and click **`Next>`**. You should see a dialog similar to this:



Enter the Maven group id, artifact id and the package name as shown. You may leave the other entries as they are for this exercise. Now click **`Finish`**. You should now find a project named **`example`** in the Package Explorer view of Eclipse that looks something like this:



This is a fully functional plugin project that can be run without modification. You can see from the file and folder entries how the values from the Maven archetype dialog affect the naming of different project elements.

Running a UI Plugin Project

Next, let us try to run the sample plugin using the test harness. Locate the test harness in the CWF distribution in a project named

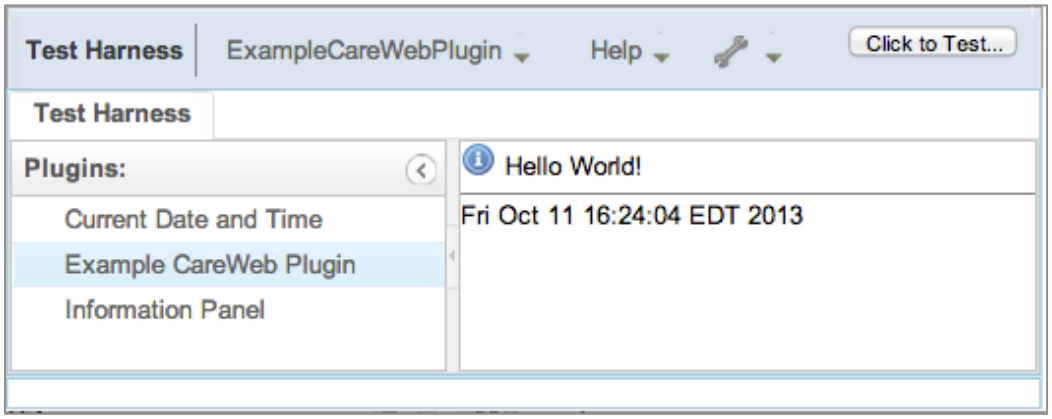
`org.carewebframework.testharness.webapp`

The test harness is a pre-configured web application project that has special logic to automatically detect plugins in the environment and display them within the UI layout. Because this default test harness implementation runs without a real back-end host system, it has fairly limited functionality, using mock services for authentication and data persistence. More capable implementations of the test harness for specific host environments are available as well.

In the test harness project, find and open the **`pom.xml`** file. In the dependencies section we need to add a reference to our sample plugin. It should look like this:

```
        .
        .
        .
<dependencies>
   <!-- Insert any plugins to be tested here. -->
   <dependency>
      <groupId>my.maven.project</groupId>
      <artifactId>example</artifactId>
      <version>0.0.1-SNAPSHOT</version>
   </dependency>
        .
        .
        .
```

Close the modified file, saving the changes you just made. It is a good idea to refresh the test harness project at this point. You are now ready to run the test harness. Locate the **`src/main/webapp/index.zul`** file in the test harness project. Right-click on the entry in the Package Explorer and select **`Run As→Run on Server`** from the context menu. If you have not already set up a web server in your Eclipse environment, you will be prompted to do so. Just follow the prompts to complete this task. Once you have set up a web server, select it from the prompt that follows and after a short delay you should see the login screen appear in the Eclipse browser (or an external browser if you have Eclipse so configured). Click the **`Login`** button (the default credentials are automatically supplied) and you should see the test harness screen. On the left pane are the plugins detected in the environment, your sample plugin among them. Select the entry for your plugin and you should see the following:



Project Structure

Now that you know how to run a plugin within the test harness, let's examine how a plugin project is structured using the example plugin that we just created. We follow the Maven convention for project structure, with folders **`src/main/java`** serving as the root folder for Java packages and code, and **`src/main/resources`** for pretty much everything else. For testing, identically named folders exist under the

src/test root. Within the **resources** folder you will find two subfolders, **META-INF** and **web**. The **META-INF** folder contains bean configuration files and property files. While this folder is also the location of the **MANIFEST.MF** file in the generated jar file, this file is not part of the project as Maven automatically produces it during the build process. The **web** folder has special significance to the ZK Framework. Resources placed under this folder may be accessed via a special URL syntax that begins with the three characters "**~./**". This is typically where web resources such as zul pages or image files are placed.

Spring Configuration – Root Profile

Let's start off by examining the Spring configuration file for our newly created plugin. Open the file **exampleCareWebPlugin-spring.xml** located in the **src/main/resources/META-INF** folder.

```
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:cwp="http://www.carewebframework.org/schema/plugin"
   xsi:schemaLocation="
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
      http://www.carewebframework.org/schema/plugin
      http://www.carewebframework.org/schema/plugin/plugin-extensions.xsd">

   <!-- This profile is processed during web server startup -->

   <beans profile="root">

      <!-- This is the sample definition for a CareWeb plugin -->
      <cwp:plugin id="exampleCareWebPlugin" name="Example CareWeb Plugin"
         url="~./my/plugin/exampleCareWebPlugin.zul">
         <cwp:resource>
            <cwp:button-resource
               caption="${labels.examplecarewebplugin.button.caption}"
               action='zscript:Messagebox.show("${labels.examplecarewebplugin.button.click.msg}");' />
            <cwp:help-resource module="exampleCareWebPluginHelp" />
            <cwp:menu-resource path="ExampleCareWebPlugin\Click Me"
               action='zscript:Messagebox.show("${labels.examplecarewebplugin.menu.click.msg}");' />
            <cwp:property-resource group="exampleCareWebPluginPropGroupName" />
         </cwp:resource>
         <!-- You may specify plugin level security constraints here:
         <cwp:security requiresAll="false">
            <cwp:authority name="PRIV_CLINICAL_PRIVILEGED_DATA_HIV_OR_AIDS"/>
            <cwp:authority name="PRIV_CLINICAL_PRIVILEGED_DATA_PSYCHIATRIC"/>
         </cwp:security>
         -->
      </cwp:plugin>

   </beans>

   <!-- This profile is processed for each managed desktop. -->

   <beans profile="desktop">

      <bean id="exampleCareWebPluginController"
         class="my.plugin.controller.ExampleCareWebPluginController"
         scope="prototype" />

   </beans>
</beans>
```

Recalling the previous discussion regarding Spring Framework integration, you will note the two profile declarations: one for **`root`** and one for **`desktop`**. Under the root profile, you will see what is labeled as the definition for our sample plugin. Note that the format is not the familiar Spring bean declaration syntax – or is it? In truth, it is a bean declaration. In fact, every declaration in a Spring configuration file must ultimately translate to a bean definition. As we noted in our previous discussion, Spring allows the use of alternative syntax for specifying a bean declaration through the use of namespace extensions. While one could specify a plugin definition using native bean syntax, the plugin namespace extension allows for a much more concise declaration.

A plugin declaration begins and ends with a **`cwp:plugin`** tag. This tag supports several attributes, some required, others optional:

cwp:plugin

Attribute	Required	Description
id	Yes	This is the unique identifier associated with the plugin. It is recommended that you choose a common prefix and use this throughout the plugin where a unique identifier is needed (bean names and package names, for example).
name	Yes	This is the display name to be associated with the plugin. While there is no absolute maximum length, it should be concise yet descriptive.
class	No	This attribute is reserved for plugins that directly extend user interface elements. It represents the implementation class.
category	No	A comma-delimited list of categories under which this plugin will appear in the designer.
copyright	No	Any copyright information associated with this plugin.
creator	No	The author of this plugin.
description	No	This is an optional description for the plugin and can be of any length.
icon	No	URL of the icon representing this plugin (for future use).
lazyLoad	No	If true (the default), the plugin is not loaded until it first becomes visible in the user interface. If false, the plugin is loaded immediately after its container is instantiated. For performance reasons, it is recommended to leave this at its default setting. However, there are circumstances where it is desirable to have a plugin load prior to becoming visible.
released	No	The date this version was released.
source	No	The source vendor of the plugin.
url	No	Though not required, custom plugins will almost always specify this attribute. This URL is the path to the zul page that represents the principal view for this component. Note the `~./` prefix. This is a special ZK notation that indicates that this is a jar-embedded web resource. It is shorthand for a resource located under the `web` folder. Using this notation, resources packaged within a jar file can be accessed by the client browser in the same fashion as a freestanding web resource.
version	No	Version # of this plugin.

A plugin declaration may also include sections for specifying associated resources, security restrictions, and serialization instructions.

The **`cwp:resource`** tag introduces the section for declaring resources to associate with the plugin. Resources may be toolbar buttons, menu items, property groups, help modules, style sheets, commands, or bean references.

Toolbar buttons are declared with the **`cwp:button-resource`** tag. These buttons appear on the common toolbar when the plugin is activated and are hidden when the plugin becomes inactive. The tag supports the following attributes:

cwp:plugin/cwp:resource/cwp:button-resource

Attribute	Required	Description
caption	Yes	This is the text that will appear on the button.
action	Yes	This is the action associated with the button that is invoked when the button is clicked. See the section on actions for more information.
icon	No	The URL of an optional image to display on the button.
tooltip	No	An optional tool tip to be displayed when the mouse hovers over the button.

The **`cwp:help-resource`** tag associates a help module with the plugin. The framework uses this information to determine which help resources to display under the help menu. See the section on the help subsystem for more information. This tag supports the following attributes:

cwp:plugin/cwp:resource/cwp:help-resource

Attribute	Required	Description
action	No	Required if no module is specified. This indicates the action to take when the menu item is clicked. See the section on actions for more information.
module	No	This is the unique identifier of the associated help module. If a module is specified, the path and action attributes will be automatically populated based on the help module definition.
path	No	Required if no module is specified. This indicates where the resource should appear under the Help menu.
topic	No	If a topic other than the default topic is to be displayed when the help content is activated, it may be specified here. Otherwise, the default topic is chosen.

The **`cwp:menu-resource`** tag declares a menu item resource. These menu items appear on the common menu when the plugin is activated and are hidden when the plugin becomes inactive. Supported attributes are:

cwp:plugin/cwp:resource/cwp:menu-resource

Attribute	Required	Description
path	Yes	This is the path of the menu item. For example, `External Resources\WebMD` would create a menu item labeled "WebMD" under a parent menu called "External Resources".
action	Yes	This is the action associated with the menu item that is invoked when the item is clicked. See the section on actions for more information.

If the plugin has an associated property group, this may be declared using the **`cwp:property-resource`** tag. A user preference plugin can use this information to determine which user preference groups to display. This tag supports a single attribute:

cwp:plugin/cwp:resource/cwp:property-resource

Attribute	Required	Description
group	Yes	This is the name of the property group to be associated with the plugin.

The **`cwp:css-resource`** tag permits the association of one or more style sheets with the plugin. Style sheets declared in this manner are managed by the framework, being loaded as required. This tag supports a single attribute:

cwp:plugin/cwp:resource/cwp:css-resource

Attribute	Required	Description
url	Yes	The URL of the style sheet.

The **`cwp:command-resource`** tag allows a plugin to bind itself to a named command (see the section on shortcut support). This allows a plugin to respond to a typed shortcut when it does not have the input focus. This tag supports a single attribute:

cwp:plugin/cwp:resource/cwp:command-resource

Attribute	Required	Description
name	Yes	The name of the associated command.

Finally, the **`cwp:bean-resource`** tag permits the association of Spring-managed beans with a plugin. Beans declared in this manner will be referenced when the associated plugin is first hosted in its container, but before the plugin is instantiated. This capability can be used to ensure that any Spring-managed beans required for the proper operation of the plugin are loaded and available. In addition, if a referenced bean implements either the **`IPluginEvent`** or the **`IPluginEventListener`** interface, it will receive plugin event notifications automatically. This tag supports the following attributes:

cwp:plugin/cwp:resource/cwp:bean-resource

Attribute	Required	Description
bean	Yes	The id of the Spring-managed bean.
required	No	If true and the bean is not found, an exception will be raised. If false and the bean is not found, the declaration is ignored. The default value is "true".

The **`cwp:security`** tag introduces the security section where access controls may be specified. This tag has one attribute:

cwp:plugin/cwp:security

Attribute	Required	Description
requiresAll	No	When true, the user must possess all named access controls in order to access the plugin. When false, possessing any one of the named access controls is sufficient. When not specified, the default value is false.

Use the **`cwp:authority`** tag within the security section to specify required security privileges or roles. This tag supports a single attribute:

cwp:plugin/cwp:security/cwp:authority

Attribute	Required	Description
name	Yes	This is the name of the security privilege or role. These follow the Spring Security naming convention where privilege names are prefixed with "PRIV_" and role names with "ROLE_".

The **`cwp:serialization`** tag introduces the section where a plugin's properties may be declared. Properties declared here can be exposed in the designer and serialized when a layout snapshot is taken. Each property is declared using the **`cwp:property`** tag, which supports several attributes:

cwp:plugin/cwp:serialization/cwp:property

Attribute	Required	Description
id	Yes	The internal name of the property.
name	Yes	The display name of the property.
config	No	Additional configuration parameters as key/value pairs. The content is type dependent.
default	No	The default value for the property.
description	No	A description of this property's function. This information is displayed by the layout designer's property editor.
editable	No	If true, the property's value appears in the designer's property editor and can be modified. If false, the property's value does not appear in the designer's property editor. The default value is true.
getter	No	The name of the property's getter method. If not specified, the name is inferred from the property's id. Specify this if a non-standard getter method is used.
serializable	No	If true, the property's value is serialized when a layout snapshot is taken. The default value is true.
setter	No	The name of the property's setter method. If not specified, the name is inferred from the property's id. Specify this if a non-standard setter method is used.
type	No	The data type of the property. The stock data types are action, boolean, choice, color, date, double, enum, icon, integer and text. Custom data types are also possible.

Spring Configuration – Desktop Profile

Next, let us examine the contents of the desktop profile, reproduced below for convenient reference:

```
<!-- This profile is processed for each managed desktop. -->

<beans profile="desktop">

   <bean id="exampleCareWebPluginController"
      class="my.plugin.controller.ExampleCareWebPluginController"
      scope="prototype" />

</beans>
```

The contents of the desktop profile will vary from project to project. In the example above there is a single bean declaration, **`exampleCareWebPluginController`**, which is marked as prototype. This is typical of beans that declare ZK controller classes, since most controller classes contain instance variables specific to the zul page they control and, therefore, require a unique controller instance to be generated for each reference. We will see how this bean reference is bound to a zul page when we examine that resource later in the section.

Note that the bean declaration uses a plugin-specific prefix (**`exampleCareWebPlugin`**) in its id. This is important to avoid naming collisions and should be used consistently across all bean declarations. If a naming collision occurs, an exception will be raised at the time the configuration file is processed unless the declaration is declared as an intentional override.

Spring Configuration – Properties

Properties are important contributors to the flexible nature of the CWF – so important that we have a detailed discussion of the different types of properties later in this document. However, because we encounter a property file in our sample plugin, the use of properties in Spring merits some treatment here.

Property references in Spring configuration files use a simple EL expression syntax. For example, consider the following bean declaration:

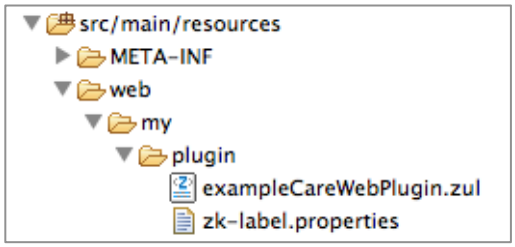
```
<bean id="sampleBean" class="test.sample.SampleBean">
       <property name="sampleProperty1" value="${test.sample.value1}" />
       <property name="sampleProperty2" value="${test.sample.value2:1234}" />
</bean>
```

Here, you see two bean properties being injected with values derived from property references. The first reference specifies no default value while the second specifies a default value of "1234". When Spring encounters a property reference that it

cannot resolve, it will use this default value or, in the absence of a default value, throw an exception. (Note: it is possible to configure the Spring IOC container to simply ignore unresolved property references, but the CWF takes the more conservative approach and requires that all property references be resolvable.) So where are property assignments found? The CWF first searches for property files in each jar's **`META-INF`** folder that match the naming pattern "***`-default.properties`**". You will find an example of this in the sample plugin you just created. Next, the CWF searches the class path for property files matching the naming pattern "**`cwf*.properties`**". As property assignments are collected from these sources, duplicate assignments overwrite previous ones. This means that a property assignment found in an external property file will replace one for the same property found in a jar-embedded property file. Thus, the precedence (from highest to lowest) for property assignments is effectively: external property file, jar-embedded property file, default value in EL expression.

Web Resources – The ZUL Page

Every UI plugin will have at least one zul page, the principal zul page, which is what gets displayed when the plugin is activated. This is the page whose URL is referenced in the **`url`** parameter of the plugin definition. Like other jar-embedded web resources, zul pages must reside under the **`web`** folder. To avoid potential naming collisions with web resources in other jar files, CWF programming convention dictates that web resources be placed under a directory tree that mirrors the package name for the project. This convention is reflected in the our sample plugin, where we chose **`my.plugin`** as our package name, as shown:



Opening the principal zul page, **`exampleCareWebPlugin.zul`**, you should see the following:

```
<?taglib uri="http://www.carewebframework.org/tld/security" prefix="sec"?>
<?taglib uri="http://www.carewebframework.org/tld/core" prefix="rc"?>

<zk xmlns="http://www.zkoss.org/2005/zul"
   xmlns:h="http://www.w3.org/1999/xhtml"
   xmlns:w="http://www.zkoss.org/2005/zk/client"
   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">

   <div id="exampleCareWebPlugin" apply="${exampleCareWebPluginController}">
      <image src="${rc:getIconPath('information.png')}"/>
      <label value="${labels.examplecarewebplugin.welcome.msg}"/>
      <separator bar="true"/>
      <label id="lblExample"/>
   </div>

</zk>
```

So our plugin zul page consists of a couple of tag library references and a div component containing four other ZK components. There are three EL style references in the page. The first, assigned to the **`apply`** attribute, represents the bean name of our controller class as we saw declared previously in the desktop profile portion of the bean configuration file. This binds our controller class, **`ExampleCareWebPluginController`**, to the **`div`** component of our zul page. Since the bean is declared with a scope of **`prototype`**, we get a new instance of the controller class each time we instantiate the page.

The next EL expression is a tag library reference where "**`rc:`**" is the prefix we have associated with the CWF core tag library in the second line of the zul page. The expression invokes the **`getIconPath`** method, which returns the full path to the named icon.

The last EL expression is a label reference. Label references represent placeholders for text that resides in a separate file, the **`zk-label.properties`** file. We discuss this further in the next section.

Web Resources – Internationalization

ZK fully supports internationalization and localization. This is the purpose of the **`zk-label.properties`** file. This file allows one to externalize the text content of an application and provide locale-specific variants. Opening this file in our sample plugin, we see the following:

```
#Example CareWeb Plugin
#UI Constants
examplecarewebplugin.name=Example CareWeb Plugin
examplecarewebplugin.button.caption=Click to Test...
examplecarewebplugin.button.click.msg=Button clicked!
examplecarewebplugin.menu.click.msg=Menu clicked!
examplecarewebplugin.menu.resource.path=Example\Example Submenu\Click Me!
examplecarewebplugin.help.resource.path=Help Contents
examplecarewebplugin.help.resource.topic=overview
examplecarewebplugin.welcome.msg=Hello World!
```

Looking back at the plugin's zul page and bean configuration file, we can see references to the placeholder values declared in this file. These occur as EL type expressions where the variable name is prefixed with "**labels.**". Thus, whenever an EL expression of this type is encountered, it will be replaced with the text value derived from this file.

So how does this support internationalization? The **zk-label.properties** file is actually just the default source for text placeholders. When resolving a text placeholder, ZK will look for content that most closely matches the current locale. It distinguishes content based on the naming of the content file, which must follow the format: **zk-label_*<language>*_*<COUNTRY>*.properties**, where ***<language>*** is the language code and ***<COUNTRY>*** is the country code. ZK will resolve placeholders from the file that is most specific to the current locale. So, for example, if the locale were French Canada, the locale id would be "**fr_CA**". ZK would attempt to resolve text placeholders by searching for content from files in this order:

1. **zk-label_fr_CA.properties**
2. **zk-label_fr.properties**
3. **zk-label.properties**

A plugin developer may choose to deliver text content for their locale only, or for multiple locales. These may be bundled within the plugin's jar file, as is the case in our example, or delivered in separate jar files where the implementer can choose which one to deploy. In addition, one may create standalone text placeholder files following the same file naming convention to override placeholder assignments. By placing these custom overrides in the **WEB-INF** folder of the web application, the implementer can tailor text content to meet their local needs.

The Main Controller

Now let us take a look at the controller class for our principal zul page:

```
public class ExampleCareWebPluginController extends PluginController {

    private static final long serialVersionUID = 1L;

    private static final Log log = LogFactory.getLog(ExampleCareWebPluginController.class);

    private Label lblExample; // This value will be injected automatically by the parent class

    @Override
    public void doAfterCompose(final Component comp) throws Exception {
        super.doAfterCompose(comp);
        log.trace("Controller composed");
        lblExample.setValue(new Date().toString());
    }

    @Override
    public void onLoad(final PluginContainer container) {
        super.onLoad(container);
    }

    @Override
    public void onUnload() {
        super.onUnload();
    }

    @Override
    public void onActivate() {
        super.onActivate();
    }

    @Override
    public void onInactivate() {
        super.onInactivate();
    }
}
```

Our controller doesn't do much at this point. As you can see, it extends the **`PluginController`** class, which ultimately extends from ZK's **`GenericForwardComposer`** class. This latter class provides auto-wiring of instance variables and event handlers (see ZK documentation for more detail). As an example of this nice feature, note the comment next to the variable named **`lblExample`** indicating that the parent class will inject (auto-wire) the value for this variable. This is possible because the variable name and type match the id and type of a ZK component in the zul page to which this controller is attached. This is why we are able to reference the label without explicitly initializing it first.

The **`doAfterCompose`** method is a common place to put initializations once the attached zul page is fully composed. It is also the place (in the super method) where the auto-wiring magic takes place.

Note the **`onXxxxx`** method overrides. These are overrides of the **`IPluginEvent`** interface methods. This is possible because the super class implements this interface. We describe this interface next.

IPluginEvent Interface

The lifecycle of a plugin consists of initial instantiation, activation and inactivation over the course of its lifetime, and destruction during application shutdown. These events are communicated via the **`IPluginEvent`** interface. A plugin requiring an awareness of one or more of these events may implement this interface. The plugin's container will detect this implementation and use it to communicate the occurrence of each event to the plugin.

onLoad

org.carewebframework.shell.plugins.IPluginEvent

Parameter	Datatype	Description
container	**PluginContainer**	**This is the container associated with the plugin. The container has useful methods that allow the plugin to communicate with its environment. The `onLoad` event provides an opportunity for the plugin to save a reference to its container.**

Signals the instantiation of the plugin. Occurs one time only during the plugin's lifecycle. For performance reasons, by default a plugin is not instantiated until the first time it is activated; therefore, it is possible that a plugin may never be instantiated.

onActivate

org.carewebframework.shell.plugins.IPluginEvent

Occurs when a plugin becomes visible. Plugins that need to defer computationally expensive operations until they become visible to the user can use this method along with the **`onInactivate`** method to determine the appropriate timing.

onInactivate

org.carewebframework.shell.plugins.IPluginEvent

Occurs when a plugin becomes hidden. This will happen, for example, if the plugin is located on a tab and the user selects a different tab.

onUnload

org.carewebframework.shell.plugins.IPluginEvent

Signals the destruction of the plugin. Occurs when the plugin is removed from the UI, either programmatically or during application shutdown. A plugin may use this event to release managed resources, for example.

IPluginEventListener Interface

An alternative to the **`IPluginEvent`** interface is the **`IPluginEventListener`** interface, which implements a more ZK-like event interface for monitoring plugin lifecycle changes by using an event object to convey event-specific information. The advantage of this interface over the **`IPluginEvent`** interface is that it can be later extended to include additional kinds of events without breaking backward compatibility. It also signals lifecycle events that are not available to the **`IPluginEvent`** interface. The **`IPluginEventListener`** interface declares only one method:

onPluginEvent org.carewebframework.shell.plugins.IPluginEventListener

Parameter	Datatype	Description
event	PluginEvent	This is the event object that describes the plugin lifecycle event being reported. See below for additional detail.

This is the callback method for reporting plugin lifecycle events. Its single parameter is an object of the **`PluginEvent`** class, described below.

PluginEvent Class

The **`PluginEvent`** class conveys information about the plugin lifecycle event.

getContainer org.carewebframework.shell.plugins.PluginEvent

Parameter	Datatype	Description
<return value>	PluginContainer	This is the container associated with the plugin. The container has useful methods that allow the plugin to communicate with its environment.

Returns the container associated with the plugin.

getAction org.carewebframework.shell.plugins.PluginEvent

Parameter	Datatype	Description
<return value>	PluginAction	An enumeration value that describes the type of lifecycle event being reported. Current values: `SUBSCRIBE` – Occurs when a listener is added as a subscriber. `LOAD` – Occurs when a plugin is initially instantiated. `ACTIVATE` – Occurs when a plugin is activated within the user interface. `INACTIVATE` – Occurs when a plugin is inactivated within the user interface. `UNLOAD` – Occurs when the plugin is being removed from its container. `UNSUBSCRIBE` – Occurs when a listener is removed as a subscriber.

Returns the action that caused the event to be fired.

Exposing Plugin Properties

A plugin may identify its accessible properties in its definition declaration. To demonstrate, we will add a property to our sample plugin. Locate and open the bean configuration file, **`exampleCareWebPlugin-spring.xml`**, from the sample plugin project. Before the **`cwp:plugin`** closing tag, add the following:

```
<cwp:serialization>
   <cwp:property name="Custom Label" id="label" type="text"
       description="This is my new custom label property." />
</cwp:serialization>
```

Properties declared in this manner may be serialized and deserialized by the Layout Manager and are accessible to the Layout Designer. Property access is delegated to the plugin's container. However, the container has no specific knowledge of the plugin it hosts and needs to be told by its plugin how a given property is to be accessed. Fortunately, the **`PluginContainer`** class has methods that allow the plugin to do just this:

registerProperty org.carewebframework.shell.plugins.PluginContainer

Parameter	Datatype	Description
instance	**Object**	**The object instance containing the property.**
propertyName	**String**	**The name of the property.**
override	**boolean**	**If the property has been previously registered, setting this to true will replace the previous registration. A value of false will retain the original registration.**

This method informs the container that the named property may be accessed via the specified object instance.

registerProperties org.carewebframework.shell.plugins.PluginContainer

Parameter	Datatype	Description
instance	**Object**	**The object instance containing the properties.**
propertyNames	**String…**	**One or more property names to register.**

This method informs the container that the named properties may be accessed via the specified object instance.

Property registration is typically done in the plugin's **`onLoad`** method. Let us modify the implementation of this method in our plugin's controller class, **`ExampleCareWebPluginController`**:

```
@Override
public void onLoad(final PluginContainer container) {
    super.onLoad(container);
    container.registerProperties(this, "label");
}
```

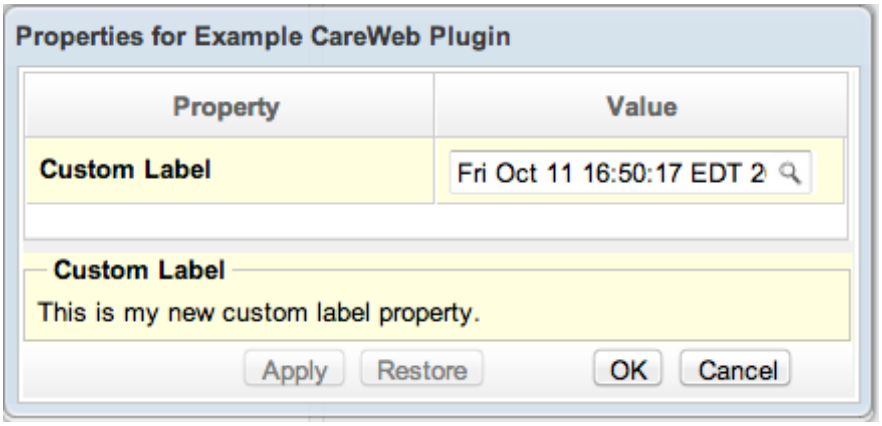
Here, we tell the container that the **`label`** property may be accessed via the plugin instance. You may wonder if the CWF can access a property before its plugin is initialized. Since a plugin's initialization is typically delayed until it is first activated, the container employs an internal proxy object to store property value assignments that occur before a plugin is initialized. Part of this initialization step is to transfer property values from the proxy objects to their actual counterparts.

We have implemented the code to register our new property, but we have not yet implemented the property itself. To do this, let's add a getter and setter method for our new **`label`** property to the controller class:

```
    public String getLabel() {
        return lblExample.getValue();
    }

    public void setLabel(String value) {
        lblExample.setValue(value);
    }
```

Now save all changes and relaunch the test harness. It should look pretty much as before. Now let's set the value of the property we just created. From the test harness main menu, click the design menu (the wrench icon) and activate design mode by checking the **`Design Mode`** menu item. Notice how the appearance of the UI changes. Right-click the sample plugin's window and from the context menu select **`Properties`**. You should see the property editor for your plugin displaying the custom property that we just created and registered:



Replace the label property's value with something of your own choosing, and then click **`OK`**. You should now see your new value in place of the original.

Writing Context-Aware Code

Each managed context declares a callback interface that extends the **`IContextEvent`** interface and is used to coordinate communication between the managed context and its subscribers (this interaction is mediated by the context manager, but more on that later). For example, the managed context for the user, **`org.carewebframework.api.context.UserContext`**, declares an **`IUserContextEvent`** callback interface, which simply extends the **`IContextEvent`** without adding anything additional. This is necessary to distinguish this callback interface from those of other managed contexts.

IContextEvent Interface

The **`IContextEvent`** interface declares the following methods:

pending org.carewebframework.api.context.IContextEvent

Parameter	Datatype	Description
silent	boolean	When true, no user interaction is permitted.
<return value>	String	If non-null/non-empty, represents a displayable reason why the context change should not proceed.

The **`pending`** method corresponds to phase one of the context change transaction. This provides a subscriber with the opportunity to veto a context change request.

committed org.carewebframework.api.context.IContextEvent

Called during the second phase of a context change transaction to signal that the change has been committed.

canceled org.carewebframework.api.context.IContextEvent

Called during the second phase of a context change transaction to signal that the change has been canceled.

Subscribing to a Context Event

An object becomes a subscriber to a shared context by simply implementing the callback interface as declared by the context of interest and registering itself with the CWF (see the section on object registration). By doing so, the object is guaranteed to be notified of context changes and have an opportunity to vote on context changes. For example, the following class implements a patient context change callback interface:

```
public class ASubscriber extends Div
   implements PatientContext.IPatientContextEvent {

   public ASubscriber() {
      FrameworkUtil.getAppFramework().registerObject(this);
   }

   @Override
   public String pending(boolean silent) {
      // Return a non-empty text string to vote no.
      return null;
   }

   @Override
   public void canceled() {
      // Called if the context change was canceled.
   }

   @Override
   public void committed() {
      // Called if the context change was committed.
   }
}
```

Note the constructor code to register the object. By doing so, the application framework will automatically connect the object instance (subscriber) to the corresponding managed context (publisher). Note also the single parameter to the **`pending`** method. If this parameter is true, your code may not interact with the user during the polling phase. In most cases, this parameter will be false. However, if the application is forcibly shutdown by, for example, a session timeout, or if the context change request originated from an external CCOW context manager, no user interaction is permitted. In addition, in such a scenario, the context change may be committed even if a no vote is received. Therefore, context change subscribers should never assume that a no vote would always result in a canceled transaction, though in most cases this will be true.

When user interaction is permitted, developers will typically use the **`pending`** method to alert the user as to the consequences of the context change and to, perhaps, elicit feedback from the user as to the desired course of action. For example, if the subscribing component contains user input that has not been committed, the user might be prompted that proceeding will result in loss of uncommitted data. One user response might be to cancel the pending context change, in which case the code would return a non-empty message string from the **`pending`** method thereby canceling the context change. Alternatively, the user could be presented with an additional option to save the changes prior to continuing with the context change, thus allowing the context change to proceed without a loss of data.

So what text value should be returned from the **`pending`** method to designate a no vote? From the context manager's standpoint, any non-empty value will do. It is recommended, however, to return an informational message indicating the reason for the no vote. If the CWF is participating in a CCOW context (not currently implemented), this text message will be returned to the CCOW context manager where it may be displayed to the user.

Typically, most developers will have little use for the **`canceled`** callback method. It must be implemented, but will most often be empty. The **`committed`** method is where the application will respond to a context change once committed, usually by updating its state to reflect the new context.

ISharedContext Interface

So how does one access the current context? Most implementers of managed contexts will provide convenience methods to do this for you (e.g., the **`UserContext`** class has a static method for retrieving the active user). Use these when available. However, if you want to do this yourself, you must first get a reference to the managed context's **`ISharedContext`** interface. This interface has the following declaration:

getContextObject org.carewebframework.api.context.ISharedContext

Parameter	Datatype	Description
pending	boolean	If true, the context object from the pending context is returned; if false, the current context is the source.
<return value>	<DomainClass>	The requested context object.

Returns the domain object associated with the current or pending context.

requestContextChange org.carewebframework.api.context.ISharedContext

Parameter	Datatype	Description
newContextObject	<DomainClass>	The object to become the new context.

Sets the specified context object into the pending context and initiates the context change transaction.

`DomainClass` is the class of the wrapped domain object. You may get a reference to the managed context's **`ISharedContext`** interface from the context manager service. The following code is from the **`UserContext`** convenience method defined for retrieving the current user context:

```
public static ISharedContext<IUser> getUserContext() {
   return (ISharedContext<IUser>)
      ContextManager.getInstance().getSharedContext(UserContext.class.getName());
}
```

Here, the **`getSharedContext`** method of the context manager service is used to locate the user context's managed context wrapper from its class name. While there are other methods for accessing managed context wrappers (e.g., by requesting them directly from the application context via their bean identifier), the method shown here is generally preferred.

Once a reference to the **`ISharedContext`** interface of the managed context has been obtained, getting a reference to the wrapped domain object is straightforward. Note the following code, again taken from a convenience method implemented by the **`UserContext`** class:

```
public static IUser getActiveUser() {
    return getUserContext().getContextObject(false);
}
```

Here, the **`getContextObject`** method of the **`ISharedContext`** interface is used to obtain a reference to the current user. Note the single Boolean argument for this method. A value of false indicates that the current (committed) domain object is being requested. A value of true would retrieve the pending domain object. A pending domain object only exists during the first phase of a context change transaction, so a value of true makes sense if used inside the **`pending`** method if one

needs to access the domain object that is yet to be committed. With either argument value, **`getContextObject`** may return a null value.

Requesting a Context Change

You may have noticed that the **`ISharedContext`** interface has a second method, **`requestContextChange`**. This method is used to initiate a context change request. Again, most implementers of managed contexts will provide a convenience method for accomplishing this. For example, the **`UserContext`** class provides the following method:

```
public static void changeUser(IUser user) {
   try {
      getUserContext().requestContextChange(user);
   } catch (Exception e) {
      log.error("Error during user context change.", e);
   }
}
```

This call initiates a request to change the context to the specified domain object, in this case a user. As we have noted, this does not guarantee that the request will be honored and the programmer should not make any assumptions about the outcome. Under most circumstances, the outcome of the request is synchronous. That is, the context change, if honored, will have been committed by the time the method returns. However, if the context change request is nested inside another context change request (for example, if a request to change the context of another shared context is made in the **`pending`** method), this will not be the case and the context change will not be committed until the enclosing context change is committed.

Status Beans

A plugin developer may wish to provide a visual indication of a plugin's enabled state. For example, if a plugin displays information for the currently selected patient and there is no patient currently selected, the plugin may want to indicate to its container that it is disabled. The container conveys this information to its enclosing UI element, which conveys it to its parent element, and so on. Many UI elements provide a visual cue when all of their child UI elements are disabled. A **`UIElementTreeView`**, for example, will gray the label of the tree node that corresponds to a disabled pane.

A problem arises when a plugin that has not yet been activated wants to convey its enabled state to its container. To address this problem, we can separate the logic that determines the plugin's enabled state from the plugin initialization logic in the form of a status bean. When a plugin's container is first initialized, it retrieves references to any bean resources declared in the resource section of its plugin definition. If any of the referenced beans implement a plugin lifecycle event handler

(**IPluginEvent** or **IPluginEventListener** interface), those beans will be notified of each lifecycle event. A status bean implements the **IPluginEventListener** callback interface to register the plugin container as an object whose enabled state is to be managed. When the status bean detects a change in the enabled state, it conveys the new state to each registered container.

So how does one create a status bean and associate it with a plugin? Let's look at a specific example. First, we must create a class that extends the **PluginStatus** abstract class and overrides its **checkDisabled** method, which should encapsulate the logic that determines whether or not a plugin's container should be disabled. Consider the following sample code that implements a status bean that updates its enabled state whenever a patient context change occurs:

```
public class StatusBean extends PluginStatus
   implements IPatientContextEvent {

    @Override
    public boolean checkDisabled() {
        final Patient patient = PatientContext.getCurrentPatient();
        return patient == null;
    }

    @Override
    public void canceled() {

    }

    @Override
    public void committed() {
        this.updateDisabled();
    }

    @Override
    public String pending(final boolean silent) {
        return null;
    }

}
```

As expected, we subclass the **PluginStatus** class, providing an implementation for the **checkDisabled** method that returns true if a patient is not selected. Our subclass also implements a patient context change event interface, **IPatientContextEvent**, so that we can detect changes in status. In the implementation of that interface's **committed** method we call the **updateDisabled** method of the superclass, which notifies registered containers of the status change.

Now that we have created an implementation of our status bean, we need to add a bean definition to the desktop profile of the Spring configuration file:

```
<bean id="myStatusBean" lazy-init="true" class="StatusBean" />
```

It is important that this bean definition be declared in the desktop profile. Note that we have opted to use the **`lazy-init`** feature. While not a requirement, there is not a need to instantiate our bean unless it actually gets used, so we elect to defer its creation. The bean is a singleton, which is not a problem since a single status bean can service multiple containers if necessary.

Next, we need to associate our status bean with its plugin. We do this in the plugin's definition entry in the root:

```
<cwp:plugin id="myPlugin" name="Sample Plugin"
   url="~./sample/MyPlugin.zul">

   <cwp:resource>
      <cwp:bean-resource bean="myStatusBean" />
   </cwp:resource>

</cwp:plugin>
```

You might wonder if referencing a desktop-scoped bean from the root scope violates the rule about a parent application context not referencing a bean from a child context. It would, except this is not truly a bean reference. Rather, we are simply saving the identifier of the status bean in the plugin's definition. This identifier is later used at the appropriate time to instantiate the status bean within the desktop scope.

That is basically all there is to implementing a status bean. One status bean can service multiple plugin's if its status determination logic meets their needs. So it isn't necessary to have a dedicated status bean for each plugin and, in fact, it is more efficient to avoid this where possible.

Writing Managed Contexts

Any object that needs to be shared across an application instance and where changes to that object must be coordinated among multiple consumers in a collaborative fashion is a candidate to be a managed context. A typical clinical application would provide managed context wrappers for several core domain objects such as user, patient, location, or encounter. While there is no specific requirement for an object to be a candidate for a managed context, there are a number of requirements imposed on the managed context wrapper itself in order for it to properly interact with the context management service. Fortunately, in nearly all cases, one may simply subclass the **`ManagedContext`** class and override a small number of methods in order to fulfill these requirements.

ManagedContext Class

The **`ManagedContext`** class hides most of the complexity of interacting with the context management service. While it is possible to write your own class to implement the required interfaces, there should rarely be a need to do so.

The context management service keeps track of all registered managed contexts (via framework registration). It has the responsibility for imposing the proper sequencing of events that must occur during a context change and properly handles nested context changes. Managed contexts delegate requested changes to their context state to the context management service and assume a completely passive role in the process after that.

A look at the class declaration of the **`ManagedContext`** class reveals some of the complexity of implementing a managed context wrapper from scratch:

```
public class ManagedContext<DomainClass> implements
   Comparable<IManagedContext>,
   IRegisterEvent,
   IManagedContext,
   ISharedContext<DomainClass>
```

This class implements a total of four interfaces. One of these, the **`IRegisterEvent`** interface, we have seen before. This signals to the CWF that an instance of this class wants to be notified every time an object is registered to the framework. This provides the managed context an opportunity to inspect each registered object for an implementation of its version of the **`IContextEvent`** callback interface that it will use to communicate with its subscribers. When an object being registered is found to implement the callback interface, the managed context automatically adds it to its list of event subscribers.

The **`Comparable`** interface implemented by the **`ManagedContext`** class is used by the context manager to sequence managed contexts in order of priority. This is used in

situations where multiple context changes are to be committed at once and a specific sequencing is desired. This occurs, for example, during logout where all managed contexts are set to null before logging out.

The **`ISharedContext`** interface, described in a previous section, is used to access the wrapped domain object and to initiate context change requests.

IManagedContext Interface

The **`IManagedContext`** interface allows the context manager to communicate with and control a managed context. It will likely only be of interest to those writing custom managed context wrappers. This interface is implemented by the **`ManagedContext`** class, which serves as a base class for managed context wrappers.

addSubscriber org.carewebframework.api.context.IManagedContext

Parameter	Datatype	Description
subscriber	**Object**	**Object that is to subscribe to context state changes.**
<return value>	**boolean**	**True if the subscription request was honored.**

Adds a context subscriber to the subscription list for this context. The candidate subscriber must implement the **`IContextEvent`** subinterface of the associated managed context or the subscription request will not be honored.

addSubscribers org.carewebframework.api.context.IManagedContext

Parameter	Datatype	Description
subscribers	**Iterable <Object>**	**An iterable list of candidate subscribers.**
<return value>	**boolean**	**True if at least one subscription request was honored.**

Issues subscription requests for multiple candidate subscribers. See **`addSubscriber`** method for restrictions.

commit org.carewebframework.api.context.IManagedContext

Parameter	Datatype	Description
accept	**boolean**	**If true, the pending change should be committed. If false, the pending change should be canceled.**

Commits or cancels a pending context change. If the **`accept`** parameter is true, the managed context should make its pending context state the current one and clear the pending state. If false, the managed context should simply clear its pending state, leaving the current context state unchanged.

getContextItems

org.carewebframework.api.context.IManagedContext

Parameter	Datatype	Description
pending	boolean	If true, the context items returned reflect the pending context state. If false, context items reflecting the committed state are returned.
<return value>	ContextItems	The set of context items reflecting the specified context state.

Returns the context items that reflect the current or pending state of the associated managed context. This can be used to serialize the context state for inclusion in a CCOW-managed context, for example. See a description of the `ContextItems` class for more details.

getContextName

org.carewebframework.api.context.IManagedContext

Parameter	Datatype	Description
<return value>	String	The subject name for the managed context.

Returns the unique name associated with this context. This corresponds to the subject name in the CCOW specification and should be unique across all managed contexts.

getContextObject

org.carewebframework.api.context.IManagedContext

Parameter	Datatype	Description
pending	boolean	If true, the pending domain object is returned; if false, the current domain object is returned.
<return value>	<DomainClass>	The wrapped domain object.

Returns the current or pending domain object from the managed context.

getPriority

org.carewebframework.api.context.IManagedContext

Parameter	Datatype	Description
<return value>	int	The priority of the managed context.

Returns the priority of the managed context. This value is used to influence sequencing of pending context change commits when more than one pending change exists.

init

org.carewebframework.api.context.IManagedContext

Initializes the managed context to its baseline state. This will typically be a null state for both the current and pending contexts.

isPending

org.carewebframework.api.context.IManagedContext

Parameter	Datatype	Description
<return value>	boolean	True if a pending context exists.

Returns true if the managed context has a pending context change.

notifySubscribers

org.carewebframework.api.context.IManagedContext

Parameter	Datatype	Description
accept	boolean	If true, the pending context has been committed. If false, the pending context has been canceled.
all	boolean	If true, all subscribers should be notified. If false, only previously polled subscribers should be notified.

Notifies subscribers of the outcome of the requested context change.

removeSubscriber

org.carewebframework.api.context.IManagedContext

Parameter	Datatype	Description
subscriber	Object	An existing subscriber.

Remove the specified subscriber from the list of subscribers. No action is taken if the specified object is not a subscriber.

removeSubscribers

org.carewebframework.api.context.IManagedContext

Parameter	Datatype	Description
subscribers	Iterable <Object>	An iterable list of current subscribers to be removed.

Removes a list of objects as subscribers. Any objects that are not current subscribers are ignored.

reset

org.carewebframework.api.context.IManagedContext

Resets a pending context by setting it to a null state. The current context is unaffected.

setContextItems

org.carewebframework.api.context.IManagedContext

Parameter	Datatype	Description
contextItems	ContextItems	Context items to be used to create a new context.
<return value>	boolean	True if a new pending context was successfully created.

Uses the supplied context items to create a new pending context.

surveySubscribers

org.carewebframework.api.context.IManagedContext

Parameter	Datatype	Description
silent	boolean	If true, the application may not in any way interact with the user during the polling process.
<return value>	String	Result of the survey. If any subscriber returns a non-empty string, that value will be returned here.

Polls all subscribers of managed contexts with pending changes via the **`pending`** method on their **`IContextEvent`** subinterface. If any subscriber returns a non-empty string, polling stops immediately and that value is returned.

Extending ManagedContext

Having reviewed the interfaces implemented by the **ManagedContext** class, one might be dissuaded from writing a managed context wrapper. However, since **ManagedContext** implements most of the required logic for you, writing a managed context wrapper is a simple matter of subclassing the **ManagedContext** class and overriding a handful of methods. The sequence of steps, using the managed context wrapper for the user domain object as an example, follows.

First, subclass the **ManagedContext** class, replacing the generic type with the domain class being wrapped:

```
public class UserContext extends ManagedContext<IUser>
```

Next, declare an inner interface that extends the **IContextEvent** callback interface. This is the interface that will be used to communicate with context change subscribers – note that you are declaring, not implementing, this interface.

```
public interface IUserContextEvent extends IContextEvent {};
```

Next, implement the constructor, calling the superclass constructor to register the subject name and callback interface. The subject name corresponds to the subject name of the CCOW specification and should be unique among all registered managed contexts.

```
public UserContext() {
   super("User", IUserContextEvent.class);
}
```

Depending on your needs, that may be all that you need to do. If you plan to support serialization of your context state, you will also need to override the **toCCOWContext** and **fromCCOWContext** methods. Review existing context wrappers to see how this is done (**UserContext** implements these methods). If you need to set a different priority for processing context change requests for your managed context, you may override the **getPriority** method and return a different value. If you need to take additional action after a context change is committed, override the **commit** method. If you do this, be certain that you call the super method as the first step. Finally, it is customary to provide static convenience methods for retrieving the current context (e.g., **getActiveUser**) and for changing the context (e.g., **changeUser**). Examples of these implementations have already been provided.

A formal description of the various context management classes and interfaces follows. The **IContextEvent** and **ISharedContext** interfaces have been described previously.

IContextManager Interface

The **`IContextManager`** interface is typically used by a managed context to notify the context manager of changes to its context state. Unless you are writing a managed context from scratch, there is probably little need to use this interface.

getContextMarshaller

org.carewebframework.api.context.IContextManager

Parameter	Datatype	Description
keyStoreName	**String**	**Name of the key store to be used for digital signature of the marshaled payload.**
<return value>	**ContextMarshaller**	**A context marshaller instance.**

The context marshaller provides a means to serialize the state of all registered contexts. The serialized payload is digitally signed to permit secure exchange of context states between separate applications.

getSharedContext

org.carewebframework.api.context.IContextManager

Parameter	Datatype	Description
className	**String**	**Fully qualified name of the managed context class.**
<return value>	**ISharedContext**	**A reference to the `ISharedContext` interface of the specified managed context.**

Returns a reference to the **`ISharedContext`** interface of the managed context whose class name is specified. If no such managed context has been registered, returns null.

localChangeBegin

org.carewebframework.api.context.IContextManager

Parameter	Datatype	Description
managedContext	**IManagedContext**	**The managed context whose pending context is about to change.**

Called to notify the context manager that a pending context state is about to be initialized.

localChangeEnd

org.carewebframework.api.context.IContextManager

Parameter	Datatype	Description
managedContext	**IManagedContext**	**The managed context whose pending context has changed.**

Called to notify the context manager that a pending context state has been initialized. Unless this is a nested context change, the context manager will begin the context change transaction sequence immediately.

Every call to **`localChangeBegin`** must have a matching call to **`localChangeEnd`**. These calls may be nested, however. The context manager does not perform any action until the **`localChangeEnd`** call corresponding to the outermost **`localChangeBegin`** call is invoked. This allows the context manager to treat nested context change requests as a single transaction. If during the polling phase any context subscriber of a nested context change returns a no vote, the entire transaction is canceled. Note that the same context may not participate in a nested context change more than once. An attempt to do so will result in an exception.

reset org.carewebframework.api.context.IContextManager

Parameter	Datatype	Description
silent	boolean	When true, no user interaction is permitted.
<return value>	boolean	True if the operation completed successfully.

Resets all managed contexts to a null state.

ContextItems Class

The **`ContextItems`** class acts as an intermediary during the serialization and deserialization of context states. Via the **`ContextItems`** class, context states are serialized into a CCOW-compliant format for conveyance to external processes such as a CCOW-compliant context manager. The task of serializing and deserializing context state is delegated to the wrapper of the respective managed context using an instance of the **`ContextItems`** class as the intermediary. During serialization, a managed context will store attributes that reflect its current state in this **`ContextItems`** instance. Similarly, during deserialization, a managed context will initialize an instance of its wrapped context using attribute values extracted from a **`ContextItems`** instance.

The serialization format employed by the **`ContextItems`** class follows the CCOW specification. A context state is represented in the form of a list of name-value pairs where each item name is prefixed with the subject name assigned to the context and optionally suffixed with a qualifier. For example, a patient context may be identified by multiple medical record numbers, each assigned by a different institution. CCOW would represent these contextual attributes for a theoretical patient as:

```
Patient.Id.MRN.Wishard_Memorial_Hospital=99468-5

Patient.Id.MRN.IU_Health_Network=384052
```

Here, in CCOW parlance, "**`Patient`**" is known as the subject label (or subject name), "**`Id`**" is the role, "**`MRN`**" is the name prefix, and the institution names represent the optional name suffix.

addItems org.carewebframework.api.context.ContextItems

Parameter	Datatype	Description
contextItems	contextItems	The context items to be added. If any of the items already exist, they are replaced by the specified values.

Adds items from one **`ContextItems`** instance to another.

addItems

org.carewebframework.api.context.ContextItems

Parameter	Datatype	Description
values	String	An encoded string of context items formatted as name=value pairs separated by line terminators. If any of the items already exist, they are replaced by the specified values.

Adds the items encoded in the **`values`** string.

clear

org.carewebframework.api.context.ContextItems

Removes all context items.

containsSubject

org.carewebframework.api.context.ContextItems

Parameter	Datatype	Description
subject	String	The subject name being sought.
<return value>	boolean	True if context items pertaining to the specified subject are present.

Returns true if any context items pertaining to the specified subject exist.

getDate

org.carewebframework.api.context.ContextItems

Parameter	Datatype	Description
itemName	String	The name of the item to be retrieved.
<return value>	Date	The date represented by the item value as a `Date` object.

Retrieves a date value associated with the named item. If no such value exists, returns null.

getIdentifier

org.carewebframework.api.context.ContextItems

Parameter	Datatype	Description
itemName	String	The name of the item to be retrieved.
sysId	String	The identifier's system id. This is appended to the item name for lookup.
<return value>	EntityIdentifier	The requested entity identifier.

Retrieves an entity identifier associated with the named item. If no such value exists, returns null.

getItem

org.carewebframework.api.context.ContextItems

Parameter	Datatype	Description
itemName	String	The name of the item to be retrieved.
<return value>	String	The value of the named item.

Retrieves a string value associated with the named item. If no such value exists, returns null.

getItem

org.carewebframework.api.context.ContextItems

Parameter	Datatype	Description
itemName	String	The name of the item to be retrieved.
suffix	String	The name suffix to append to the item name.
<return value>	String	The value of the named item.

Retrieves a string value associated with the item specified by the item name and suffix.

getItem

org.carewebframework.api.context.ContextItems

Parameter	Datatype	Description
itemName	String	The name of the item to be retrieved.
clazz	Class <T>	Class of item to be retrieved.
<return value>	T	An instance of the specified class.

Returns the named item as an instance of the specified class. The class must have a context serializer registered for it. If no item with the specified item name exists, null is returned.

getItemNames

org.carewebframework.api.context.ContextItems

Parameter	Datatype	Description
<return value>	Set <String>	A set that represents all of the item names in the context items collection.

Provides access to all of the item names contained within the context items collection.

getSuffixes

org.carewebframework.api.context.ContextItems

Parameter	Datatype	Description
prefix	String	The name prefix being sought.
<return value>	Map <String, String>	A map of suffixes found within items matching the specified prefix. The map key is the suffix itself and the map value is the value associated with the item.

Provides a means to access all suffixed forms of a given context item. For the aforementioned theoretical patient, a call to **`getSuffixes`** passing a prefix value of **`Patient.Id.MRN`** would return a map with two entries. The map keys would be **`Wishard_Memorial_Hospital`** and **`IU_Health_Network`** with values of **`99468-5`** and **`384052`**, respectively.

removeSubject

org.carewebframework.api.context.ContextItems

Parameter	Datatype	Description
subject	String	The name of the subject whose related items are to be removed.

Removes all context items associated with the specified subject.

setDate org.carewebframework.api.context.ContextItems

Parameter	Datatype	Description
itemName	String	The name of the item whose value is to be set.
date	Date	The value of the named item to the CCOW-encoded form of the specified date. If null, any existing value for the named item is removed.

Sets a named context item to a date value. If the named context item already exists, its current value will be replaced.

setIdentifier org.carewebframework.api.context.ContextItems

Parameter	Datatype	Description
itemName	String	The name of the item whose value is to be set.
value	EntityIdentifier	The entity identifier to set. If null, any existing value for the named item is removed.

Sets a named context item to an entity identifier value. If the named context item already exists, its current value will be replaced.

setItem org.carewebframework.api.context.ContextItems

Parameter	Datatype	Description
itemName	String	The name of the item whose value is to be set.
value	String	The new value for the named item. If null, any existing value for the named item is removed.

Sets a named context item to a string value. If the named context item already exists, its current value will be replaced.

setItem org.carewebframework.api.context.ContextItems

Parameter	Datatype	Description
itemName	String	The name of the item whose value is to be set.
value	String	The new value for the named item. If null, any existing value for the named item is removed.
suffix	String	A name suffix qualifier.

Sets the context item specified by the given item name qualified by a name suffix to a string value. If the named context item already exists, its current value will be replaced.

setItem org.carewebframework.api.context.ContextItems

Parameter	Datatype	Description
itemName	String	The name of the item whose value is to be set.
object	Object	Sets the value of the named item to the CCOW-encoded form of the specified object. If null, any existing value for the named item is removed.

Sets a named context item to a value. The class of the value must have a context serializer associated with it. If the named context item already exists, its current value will be replaced.

toString org.carewebframework.api.context.ContextItems

Parameter	Datatype	Description
<return value>	String	The serialized form of the context item collection.

Returns the serialized form of the context item collection as a list of name-value pairs delimited by line terminators.

IContextSerializer Interface

The **`IContextSerializer`** interface allows extending data type support for serializing and deserializing additional object types. This is accomplished by creating an implementation of the interface for a specific target class and registering it with the Context Serializer Registry. The **`IContextSerializer`** interface declares the following methods:

deserialize org.carewebframework.api.context.IContextSerializer

Parameter	Datatype	Description
value	String	The serialized form of the object.
<return value>	Object	The deserialized object.

Deserializes an object.

getType org.carewebframework.api.context.IContextSerializer

Parameter	Datatype	Description
<return value>	Class	The target class supported by this context serializer.

Returns the class to be serialized and deserialized.

serialize org.carewebframework.api.context.IContextSerializer

Parameter	Datatype	Description
object	Object	The object to be serialized.
<return value>	String	The serialized form of the object.

Serializes an object.

To register an implementation of the **`IContextSerializer`** interface, simply declare it in the root profile of your Spring configuration file. For example,

```
<bean class="org.acme.cwf.api.context.NameSerializer" />
```

The Context Serializer Registry will automatically detect and register it.

Writing Custom UI Elements

For most development needs the usual mechanism for authoring UI plugins, as described earlier, will suffice. As we saw, the **`UIElementPlugin`** class, a subclass of **`UIElementZKBase`**, wraps an instance of the **`PluginContainer`** class, which hosts the plugin. The **`UIElementPlugin`** wrapper provides the necessary abstraction between the Layout Manager and the plugin, with the **`PluginContainer`** acting as the intermediary. Occasionally, however, a need may arise where the standard plugin will not do. For example, you may want to create a new UI element that can host multiple child elements. The standard plugin cannot do this. In this case, writing a custom UI element becomes necessary. To do this, you will most often subclass the **`UIElementZKBase`** class, though it is certainly possible and permissible to start with other descendants of **`UIElementBase`**. Writing a custom UI element requires, at a minimum:

- Creating a subclass of **`UIElementBase`** or one of its descendants (most often **`UIElementZKBase`**).
- Creating a Spring configuration file containing the plugin definition. This is similar to the declaration for a standard CWF UI plugin.

To illustrate how to go about creating a custom UI element, we will present two examples from stock UI elements provided with the CWF distribution. The first is a simple UI element that wraps the ZK **`Button`** component and exposes some of its properties. The second example, wrapping the ZK **`Tabbox`** and related components, is slightly more complicated and is provided to illustrate how to go about creating a custom UI element that can host multiple child elements (in this case, tabs).

Let's consider the simple example where we want to create a UI element that wraps a ZK **`Button`** component. The **`UIElementButton`** class does just this. Like a standard CWF plugin, it has a Spring configuration file located in the **`META-INF`** resource folder. The plugin definition declaration looks like this:

```
<cwp:plugin id="button"
   category="Stock Objects"
   class="org.carewebframework.shell.layout.UIElementButton"
   name="${labels.cwf.shell.plugin.button.name}"
   description="${labels.cwf.shell.plugin.button.description}">
   <cwp:serialization>
      <cwp:property id="label"
         default="${labels.cwf.shell.plugin.button.label.default_}"
         name="${labels.cwf.shell.plugin.button.label.name}"
         description="${labels.cwf.shell.plugin.button.label.description}" />
      <cwp:property id="icon" type="icon"
         name="${labels.cwf.shell.plugin.button.icon.name}"
         description="${labels.cwf.shell.plugin.button.icon.description}" />
      <cwp:property id="hint"
         name="${labels.cwf.shell.plugin.button.hint.name}"
         description="${labels.cwf.shell.plugin.button.hint.description}" />
      <cwp:property id="action" type="action"
         name="${labels.cwf.shell.plugin.button.action.name}"
         description="${labels.cwf.shell.plugin.button.action.description}" />
   </cwp:serialization>
</cwp:plugin>
```

Note the similarity to the declaration for a standard plugin. The essential difference is the **`url`** attribute has been replaced by a **`class`** attribute. The **`class`** attribute identifies the implementing class for the UI element and must be a descendant of **`UIElementBase`**. Other tags and attributes are the same as we have seen before.

The implementation of the **`UIElementButton`** class looks like this:

```
public class UIElementButton extends UIElementActionBase {

    static {
        registerAllowedParentClass(UIElementButton.class, UIElementBase.class);
    }

    private final Button button = new Button();

    public UIElementButton() {
        super();
        setOuterComponent(button);
    }

    @Override
    public String getInstanceName() {
        return getDisplayName() + " (" + getLabel() + ")";
    }

    /**
     * Returns the button's label text.
     *
     * @return The label text.
     */
    public String getLabel() {
        return button.getLabel();
    }

    /**
     * Sets the button's label text.
     *
     * @param value The label text.
     */
    public void setLabel(String value) {
        button.setLabel(value);
    }

    /**
     * Sets the URL of the icon to be display on the button.
     *
     * @param url Icon URL.
     */
    public void setIcon(String url) {
        button.setImage(url);
        button.invalidate();
    }

    /**
     * Returns the URL of the icon to be display on the button.
     *
     * @return Icon URL.
     */
    public String getIcon() {
        return button.getImage();
    }

}
```

From this listing one can see that much of the code is dedicated to exposing properties of the underlying ZK component. These properties correspond to the property declarations in the preceding configuration file and allow the Layout Manager to access them for purposes of serialization and deserialization.

Focusing on the remaining code, one can see that the class initializes the wrapped ZK **`Button`** component and the constructor registers it via the **`setOuterComponent`**

method. Since this UI element accepts no children and wraps only a single ZK component, there is no need to call the **setInnerComponent** method, though doing so would not cause problems. In fact, calling only one of these methods has the same outward effect of calling both with the same argument. In other words, setting only the outer or inner component effectively sets the other.

Note also the overridden method **getInstanceName**. This method is used by the Layout Designer to display a friendly name for an instance of the UI element. The purpose of the override is to provide additional information to easily distinguish among multiple instances of the same UI element. In this case, we are adding the button's label text to the name. This override is entirely optional, but often desirable.

Finally, note that there is a static initializer that calls **registerAllowedParentClass**. This tells the Layout Manager that this UI element will accept any descendant of **UIElementBase** as a parent. Since we don't register any child classes, the UI element will not accept any children.

As one can see, wrapping a simple ZK component is a straightforward task, requiring minimal programming. Wrapping a more complex ZK component, like the **Tabbox**, for example, is a bit more involved. Recall that the ZK **Tabbox** component requires two child components, **Tabs** and **Tabpanels**, which in turn hold multiple instances of **Tab** and **Tabpanel** components, respectively. Together, a **Tab** component paired with a **TabPanel** component comprises a single tab in the UI. Wrapping a composite component like this, where there is a root component with multiple children, is best accomplished by creating two UI elements, one for the root container and one for the child elements. The CWF has several examples of this:

- **UIElementSplitterView** / **UIElementSplitterPane**
- **UIElementTabView** / **UIElementTabPane**
- **UIElementTabMenu / UIElementTabMenuPane**
- **UIElementTreeView** / **UIElementTreePane**
- **UIElementStepView** / **UIElementStepPane**

Here we will focus on the **UIElementTabView** and **UIElementTabPane** implementations. Like many of the UI elements in this group, at most only one of the child UI elements will be visible at any given time, depending of the state of the associated selector (in this case, the tab, but it could just as easily be a button in a step view or a tree node in a tree view).

The plugin definition declaration for **UIElementTabView** looks like this:

```
<cwp:plugin id="tabview" category="Stock Objects"
   class="org.carewebframework.shell.layout.UIElementTabView"
   name="${labels.cwf.shell.plugin.tabview.name}"
   description="${labels.cwf.shell.plugin.tabview.description}">
   <cwp:serialization>
      <cwp:property id="orientation" type="choice"
         name="${labels.cwf.shell.plugin.tabview.orientation.name}"
         description="${labels.cwf.shell.plugin.tabview.orientation.description}">
         <cwp:config>
            <cwp:entry key="values">horizontal,vertical</cwp:entry>
         </cwp:config>
      </cwp:property>
      <cwp:property id="color" type="color"
         name="${labels.cwf.shell.plugin.tabview.color.name}"
         description="${labels.cwf.shell.plugin.tabview.color.description}" />
      <cwp:property id="" type="tabs"
         name="${labels.cwf.shell.plugin.tabview.tabs.name}"
         description="${labels.cwf.shell.plugin.tabview.tabs.description}" />
   </cwp:serialization>
</cwp:plugin>
```

The only surprise here is the last property declaration with an empty id and an unfamiliar property type, "**`tabs`**". The CWF Designer natively supports editors for standard property types like text, integer, Boolean, enumerations, color, etc. One may extend support for additional property types through a custom property editor. The "**`tabs`**" property type is such an example. We discuss how to write a custom property editor in the next section. The property declaration lacks an id because the underlying implementation doesn't expose it with a getter and setter method, so there is no real property to which to bind. Rather, the custom property editor manages the necessary state changes directly. We call properties of this kind "pseudo-properties".

The implementation of **`UIElementTabView`** looks like this:

```
public class UIElementTabView extends UIElementZKBase {
    static {
        registerAllowedParentClass(UIElementTabView.class, UIElementBase.class);
        registerAllowedChildClass(UIElementTabView.class, UIElementTabPane.class);
        PropertyTypeRegistry.register("tabs", null, PropertyEditorTabView.class);
    }

    private final Tabbox tabBox;

    private UIElementTabPane activePane;

    public UIElementTabView() throws Exception {
        super();
        maxChildren = Integer.MAX_VALUE;
        tabBox = (Tabbox) createFromTemplate();
        setOuterComponent(tabBox);
        tabBox.setSclass("cwf-tabbox");
        tabBox.addEventListener(Events.ON_SELECT, new EventListener<SelectEvent<?, ?>>() {

            @Override
            public void onEvent(SelectEvent<?, ?> event) throws Exception {
                setActivePane((UIElementTabPane) getAssociatedUIElement(event.getTarget()));
            }

        });
    }

    /**
     * Sets the orientation which can be horizontal or vertical.
     */
    public void setOrientation(String orientation) {
        tabBox.setOrient(orientation);
    }

    /**
     * Returns the orientation (horizontal or vertical).
     */
    public String getOrientation() {
        return tabBox.getOrient();
     }

    /**
     * Need to detach both the tab and the tab panel of the child component.
     */
    @Override
    protected void beforeRemoveChild(UIElementBase child) {
        if (child == activePane) {
            setActivePane(null);
        }
    }

    /**
     * Sets the active (visible) pane.
     */
    protected void setActivePane(UIElementTabPane pane) {
        if (activePane != null) {
            activePane.activate(false);
        }

        activePane = pane;

        if (activePane != null) {
            activePane.activate(true);
        }
    }

    /**
     * Overrides activateChildren to ensure that only the active pane is affected.
     */
    @Override
    public void activateChildren(boolean activate) {
        if (activePane == null) {
            activePane = (UIElementTabPane) getAssociatedUIElement(tabBox.getSelectedTab());
        }

        if (activePane != null) {
            activePane.activate(activate);
        }
    }
}
```

First, note the static initializer. It calls **registerAllowedParentClass** to indicate that it will accept any descendant of **UIElementBase** as a parent. It then calls **registerAllowedChildClass** to indicate that it will only accept an instance of **UIElementTabPane** as a child. An attempt to add any other child will be rejected. Finally, it registers its custom property editor via a call to **PropertyTypeRegistry.register** (more on this in the next section).

Next, the constructor initializes the wrapped ZK components: a **Tabbox** and its **Tabs** and **Tabpanels** children. It does this by way of a call to the **createFromTemplate** method, which creates ZK components based on a zul page. This is a convenient way to create a composite UI element, but this could also have been done completely in code. The constructor also sets the outer wrapped component as the **Tabbox** itself. Note that it does not set an inner component. That is because here there are essentially two inner components, **Tabs** and **Tabpanels** and the default logic for adding a child to a single inner component won't work. Finally, the constructor adds an event listener to the **Tabbox** component to capture the **onSelect** event so that it can activate or inactivate its children based on the user's selection. Note the use of the **getAssociatedUIElement** call. This enables going from the ZK component (in this case, the newly selected tab) to the UI element that wraps it (the associated **UIElementTabPane**).

Finally, we override two key methods, **activateChildren** and **beforeRemoveChild**. Recall that the **UIElementTabView** allows at most one child to be active (visible) at a time. Because the base class does not support this behavior, we must track which child is currently active and ensure that only that child receives an activation request. We override **activateChildren** to limit the propagation of an activation request to the active tab pane. We override **beforeRemoveChild** to take special action if the child UI element being removed is the active pane.

Next, let's take a look at the child UI element, **UIElementTabPane**. Its plugin definition declaration looks like this:

```
<cwp:plugin id="tabpane" category="Stock Objects"
   class="org.carewebframework.shell.layout.UIElementTabPane"
   name="${labels.cwf.shell.plugin.tabpane.name}"
   description="${labels.cwf.shell.plugin.tabpane.description}">
   <cwp:serialization>
      <cwp:property id="label"
         default="${labels.cwf.shell.plugin.tabpane.label.default_}"
         name="${labels.cwf.shell.plugin.tabpane.label.name}"
         description="${labels.cwf.shell.plugin.tabpane.label.description}" />
      <cwp:property id="hint"
         name="${labels.cwf.shell.plugin.tabpane.hint.name}"
         description="${labels.cwf.shell.plugin.tabpane.hint.description}" />
      <cwp:property id="color" type="color"
         name="${labels.cwf.shell.plugin.tabpane.color.name}"
         description="${labels.cwf.shell.plugin.tabpane.color.description}" />
   </cwp:serialization>
</cwp:plugin>
```

There are no real surprises here. The implementation of **`UIElementTabPane`** looks like this:

```
public class UIElementTabPane extends UIElementZKBase {
    static {
        registerAllowedParentClass(UIElementTabPane.class, UIElementTabView.class);
        registerAllowedChildClass(UIElementTabPane.class, UIElementBase.class);
    }

    private final Tab tab = new Tab();

    private final Caption caption = new Caption();

    private final Tabpanel tabPanel = new Tabpanel();

    /**
     * Set up the tab and tab panel ZK components. Note that we use a custom widget override to
     * allow setting the color of the caption text.
     */
    public UIElementTabPane() {
        super();
        setOuterComponent(tabPanel);
        associateComponent(tab);
        tabPanel.setSclass("cwf-tab-panel");
        tabPanel.setHeight("100%");
        tab.setSclass("cwf-tab");
        tab.setWidgetOverride(CUSTOM_COLOR_OVERRIDE,
            "function(value) {jq(this).find('.z-tab-text').css('color',value?value:'');}");
        tab.appendChild(caption);
        caption.setSclass("cwf-tab-caption");
    }

    /**
     * Make this tab pane active.
     */
    @Override
    public void bringToFront() {
        super.bringToFront();
        ((UIElementTabView) getParent()).setActivePane(this);
    }

    /**
     * Requires moving both ZK components.
     */
    @Override
    protected void afterMoveTo(int index) {
        moveChild(tab, index);
        moveChild(tabPanel, index);
    }

    /**
     * The caption label is the instance name.
     */
    @Override
    public String getInstanceName() {
        return getLabel();
    }

    /**
     * Sets the visibility of the tab and tab panel.
     */
    @Override
    protected void updateVisibility(boolean visible, boolean activated) {
        tab.setSelected(activated);
        tab.setVisible(visible);
    }
```

```
    /**
     * Apply/remove the design context menu both tab and tab panel.
     *
     * @param contextMenu The design menu if design mode is activated, or null if it is not.
     */
    @Override
    protected void setDesignContextMenu(Menupopup contextMenu) {
        setDesignContextMenu(tabPanel, contextMenu);
        setDesignContextMenu(tab, contextMenu);
    }

    /**
     * Apply the disable style when a tab is disabled.
     */
    @Override
    public void setEnabled(boolean enabled) {
        super.setEnabled(enabled);
        tab.setSclass(enabled ? "cwf-tab" : "cwf-tab-disabled");
    }

    /**
     * Applies color to the tab caption text as well as the tab panel.
     */
    @Override
    protected void applyColor() {
        super.applyColor();
        applyColor(tab);
        tabPanel.invalidate();
    }

    @Override
    protected void bind() {
        Tabbox tabbox = (Tabbox) getParent().getOuterComponent();
        tabbox.getTabs().appendChild(tab);
        tabbox.getTabpanels().appendChild(tabPanel);
    }

    @Override
    protected void unbind() {
        tab.detach();
        tabPanel.detach();
    }

    /*package*/Caption getCaption() {
        return caption;
    }

    /**
     * Returns the caption label.
     *
     * @return
     */
    public String getLabel() {
        return caption.getLabel();
    }

    /**
     * Sets the caption label.
     *
     * @param value
     */
    public void setLabel(String value) {
        tab.setLabel(value);
        caption.setLabel(value);
    }

    /**
     * Hint text should be applied to the tab.
     */
    @Override
    protected void applyHint() {
        tab.setTooltiptext(getHint());
    }
}
```

Here, the static initializer informs the Layout Manager that it will accept only an instance of **UIElementTabView** as a parent, but will accept any **UIElementBase** instance as a child. Remembering that a tab in ZK really consists of two separate components, a **Tab** and a **Tabpanel**, **UIElementTabPane** creates and initializes an instance of each. In the constructor, it sets the **Tabpanel** as its outer (and implicitly its inner) component. It also calls **associateComponent** to associate itself with the **Tab** component (remember, this association is necessary to allow the **onSelect** event handler in the **UIElementTabView** class to derive the **UIElementTabPane** associated with the ZK **Tab** component that was selected). It is worth noting that the **setInnerComponent** and **setOuterComponent** methods call **associateComponent** internally.

Ignoring the getter and setter methods for the two properties, there are several method overrides that merit further examination. As with the **UIElementTabView** class, we override the **getInstanceName** method to return more meaningful display text for the Layout Designer. The **afterMoveTo** method override is necessary because we need to manipulate two ZK components when the UI element is moved relative to its siblings. The default implementation for **afterMoveTo** manipulates only the outer component. For the same reason, we override the **bind, unbind**, and **setDesignContextMenu** methods. We also must override **updateVisibility** to set the visibility and selection state for the tab. The **setEnabled** override modifies the visual appearance of the Tab component (by changing its style) to reflect the enabled state.

That is essentially all there is to creating a custom UI element.

Writing Custom Property Editors

In the implementation of the **`UIElementTabView`** class, we introduced the concept of custom property editors. The CWF provides stock property editors for common data types: action, boolean, choice, color, date, double, enum, icon, integer and text. For manipulating property settings for other data types, you must implement a custom property editor and then register it. A custom property editor must be a descendant of the abstract class **`PropertyEditorBase`**. Depending on your requirements, you may subclass **`PropertyEditorBase`** directly, or use one of its subclasses:

- **`PropertyEditorCustom`** wraps a ZK Bandbox that can be customized to support editing simple or complex data types.
- **`PropertyEditorCustomTree`** subclasses **`PropertyEditorCustom`** to provide a basis for property editors that manipulate child UI elements using a tree view representation. The tab editor for the **`UIElementTabView`** class is an example of this.
- **`PropertyEditorList`** wraps a ZK **`Combobox`** to provide support for property settings based on a fixed list of possible values.

The procedure for creating a custom property editor requires at a minimum the following two steps:

- Subclass one of the existing property editor classes to create the implementation.
- Register the property editor class to the Property Type Registry.

To illustrate this process, let's look at the implementation of two very different property editors: one for editing simple text, the other for more complex manipulation of tabs within a tab view.

First, we will look at the implementation for the property editor for editing integer values, the **`PropertyEditorInteger`** class. This class subclasses **`PropertyEditorBase`** directly and has the following implementation:

```
public class PropertyEditorInteger extends PropertyEditorBase {

    private final Intbox intbox;

    public PropertyEditorInteger() {
        super(new Intbox());
        intbox = (Intbox) component;
    }

    @Override
    protected void init(UIElementBase target, PropertyInfo propInfo, PropertyGrid propGrid) {
        super.init(target, propInfo, propGrid);
        intbox.setMaxlength(9);
        intbox.addForward(Events.ON_CHANGING, propGrid, Events.ON_CHANGE);
        Integer min = propInfo.getConfigValueInt("min", null);
        Integer max = propInfo.getConfigValueInt("max", null);

        if (min != null || max != null) {
            SimpleSpinnerConstraint constraint = new SimpleSpinnerConstraint();
            constraint.setMin(min);
            constraint.setMax(max);
            intbox.setConstraint(constraint);
        }
    }

    @Override
    protected String getValue() {
        return intbox.getText();
    }

    @Override
    protected void setValue(Object value) {
        intbox.setText((String) value);
        updateValue();
    }
}
```

Here, the constructor calls the super constructor, passing it an instance of the ZK component that will be hosted in the property grid of the Layout Designer, in this case an **`Intbox`**. For convenience, it saves a reference to the **`Intbox`** in an instance variable.

The **`init`** method may be overridden, as it is here, to provide any additional initialization that may be required after an instance of the property editor is bound to a property of a UI element and added to the property grid. The base implementation of this method forwards **`onChange`** and **`onSelect`** events from the ZK component of the property editor to the property grid, allowing the property grid to change its state based on these events. Here, we also forward the **`onChanging`** event to the property grid. This allows the property grid to respond as the content of the **`Intbox`** is changed. We also pull two additional parameters, the minimum and maximum allowable value, from the configuration parameters associated with the property definition and apply these constraints to the **`Intbox`**.

We also provide required implementations for the abstract methods, **`getValue`** and **`setValue`**. As expected, these methods allow the property grid to get or set the current value of the property being edited. Note that the **`setValue`** implementation

calls **`updateValue`** to record the new value of the property. This allows the property editor to later determine if a value has changed from the previous one.

The final step requires that we register the **`PropertyEditorInteger`** class with the Property Type Registry:

```
PropertyTypeRegistry.register("integer", int.class, PropertyEditorInteger.class);
```

This tells the property type registry that we a registering a property type called "**`integer`**" that manipulates the **`int`** class using the property editor implementation of **`PropertyEditorInteger`**.

Next, let's look at the implementation of a property editor for a more complex data type. The implementation for the tab editor used by the **`UIElementTabView`** class is surprisingly concise:

```
public class PropertyEditorTabView
    extends PropertyEditorCustomTree<UIElementTabPane> {

    public PropertyEditorTabView() throws Exception {
        super(UIElementTabPane.class, "label", false);
    }
}
```

This is possible because the superclass, **`PropertyEditorCustomTree`**, handles most of the complexity of manipulating the child UI elements. The constructor simply calls the inherited constructor, passing three parameters. The first parameter is the UI element class that is to be manipulated by the property editor. In this case, the tab property editor manipulates instances of **`UIElementTabPane`**. The second is the name of the property on the target UI element that will be synchronized with the label of the corresponding tree node. The editor will synchronize the tree view to changes in this property's value, and vice versa. In this case, it is the **`label`** property that is specified. This means that the tree view will display a list of tabs using their label property and that any changes to the label name will remain synchronized. The final parameter indicates that the associated tree view is not hierarchical (i.e., it is a single level only).

Recall from the previous section that the **`UIElementTabView`** class registered the custom editor in its static initializer. Here again is the code that does this:

```
PropertyTypeRegistry.register("tabs", null, PropertyEditorTabView.class);
```

The first parameter is the name by which the property type is to be referenced in the plugin definition. This must be unique across all registered property editors. The second is the Java class that represents the data type the editor will manipulate.

For custom editors that manipulate property values directly, a value of null is appropriate. The final parameter is the implementation class of the property editor.

For additional examples of custom property editors, we invite you to explore the numerous implementations that can be found in the **`org.carewebframework.shell.designer`** package.

Security

The CWF provides an abstraction layer for security services. This allows for various security implementations to be employed. The CWF distribution provides an abstract implementation based on Spring Security that may be extended to expose authentication and authorization services provided by the underlying host system. While Spring Security is not required and alternative security implementations are possible, the CWF security service does follow the Spring Security convention for naming security authorities. A security authority may be a role or an individual privilege. These are distinguished by the presence of a special prefix: "`ROLE_`" for a role-based authority and "`PRIV_`" for a privilege.

ISecurityService Interface

The `ISecurityService` interface provides access to the underlying security implementation.

canChangePassword

org.carewebframework.api.security.ISecurityService

Parameter	Datatype	Description
<return value>	boolean	True if the user may change his/her password.

Returns whether the current user may change his/her password.

changePassword

org.carewebframework.api.security.ISecurityService

Invokes the dialog for changing a user's password.

changePassword

org.carewebframework.api.security.ISecurityService

Parameter	Datatype	Description
oldPassword	String	The current user password.
newPassword	String	The new user password.
<return value>	String	Null or empty if the operation succeeded. Otherwise, it is a displayable reason why attempt failed.

Changes the current user's password.

generateRandomPassword

org.carewebframework.api.security.ISecurityService

Parameter	Datatype	Description
<return value>	String	A randomly generated password.

Generates a random password, observing the password naming rules of the underlying security system.

getAuthenticatedUser

org.carewebframework.api.security.ISecurityService

Parameter	Datatype	Description
<return value>	IUser	The currently authenticated user, or null if authentication has not yet occurred.

Returns the currently authenticated user as an `IUser` object. If authentication has not yet occurred, the method returns null.

hasDebugRole

org.carewebframework.api.security.ISecurityService

Parameter	Datatype	Description
<return value>	boolean	True if the current user has been granted debug mode privilege.

Returns true if the currently authenticated user has been granted debug mode privilege. This may be used to conditionally expose debugging capabilities in the UI.

isAuthenticated

org.carewebframework.api.security.ISecurityService

Parameter	Datatype	Description
<return value>	boolean	True if the current execution has been authenticated.

Returns whether or not the current execution has been authenticated.

isGranted

org.carewebframework.api.security.ISecurityService

Parameter	Datatype	Description
authority	String	The name of the authority to check. This may be a role or a privilege.
<return value>	boolean	True if the current user has been granted the authority.

Returns whether or not the current user has been granted the specified authority.

isGranted

org.carewebframework.api.security.ISecurityService

Parameter	Datatype	Description
authorities	String	A comma-delimited list of authorities to check. These may be roles and/or privileges.
requiresAll	boolean	If true, all of the listed authorities must be granted. If false, at least one of the listed authorities must be granted.
<return value>	boolean	True if the current user has been granted the required authorities.

Returns whether or not the current user has been granted the required authorities.

loginDisabled

org.carewebframework.api.security.ISecurityService

Parameter	Datatype	Description
<return value>	String	If logins have been disabled, this will be an informational message explaining the reason. If logins have not been disabled, this will be null.

Returns a displayable information message when logins have been disabled, or null if logins have not been disabled.

logout org.carewebframework.api.security.ISecurityService

Parameter	Datatype	Description
force	boolean	If true, force logout without user interaction.
target	String	The target url for subsequent login. May be null.
message	String	A message to indicate reason for logout. May be null.
<return value>	boolean	True if the logout was successful.

Log out the current desktop.

setAuthorityAlias org.carewebframework.api.security.ISecurityService

Parameter	Datatype	Description
authority	String	An authority name.
alias	String	An alias for the specified authority. If null, any existing alias is removed.

Adds (or removes) an alias for an existing authority. This is useful in situations where the CWF may use one authority name to grant access to a feature, but the underlying host system may use a different name.

validatePassword org.carewebframework.api.security.ISecurityService

Parameter	Datatype	Description
password	String	The password to be validated.
<return value>	boolean	True if the password is valid.

Returns true if the password matches the current user's password.

SecurityUtil Class

The **`SecurityUtil`** class provides several static convenience methods for accessing the security services and its methods.

getAuthenticatedUser [S] org.carewebframework.api.security.SecurityUtil

Parameter	Datatype	Description
<return value>	IUser	The currently authenticated user, or null if authentication has not yet occurred.

Returns the currently authenticated user as an **`IUser`** object. If authentication has not yet occurred, the method returns null.

getAuthenticatedUsername [S] org.carewebframework.api.security.SecurityUtil

Parameter	Datatype	Description
<return value>	String	The username of the currently authenticated user, or null if authentication has not yet occurred.

Returns the username of the currently authenticated user. If authentication has not yet occurred, the method returns null.

getSecurityService [S]

org.carewebframework.api.security.SecurityUtil

Parameter	Datatype	Description
<return value>	ISecurityService	A reference to the security service.

Returns a reference to the active security service.

hasDebugRole [S]

org.carewebframework.api.security.SecurityUtil

Parameter	Datatype	Description
<return value>	boolean	True if the current user has been granted debug mode privilege.

Returns true if the currently authenticated user has been granted debug mode privilege. This may be used to conditionally expose debugging capabilities in the UI.

isAuthenticated [S]

org.carewebframework.api.security.SecurityUtil

Parameter	Datatype	Description
<return value>	boolean	True if the current execution has been authenticated.

Returns whether or not the current execution has been authenticated.

isGranted [S]

org.carewebframework.api.security.SecurityUtil

Parameter	Datatype	Description
authority	String	The name of the authority to check. This may be a role or a privilege.
<return value>	boolean	True if the current user has been granted the authority.

Returns whether or not the current user has been granted the specified authority.

isGrantedAll [S]

org.carewebframework.api.security.SecurityUtil

Parameter	Datatype	Description
authorities	String	A comma-delimited list of authorities to check. These may be roles and/or privileges.
<return value>	boolean	True if the current user has been granted all of the listed authorities.

Returns whether or not the current user has been granted all the listed authorities.

isGrantedAny [S]

org.carewebframework.api.security.SecurityUtil

Parameter	Datatype	Description
authorities	String	A comma-delimited list of authorities to check. These may be roles and/or privileges.
<return value>	boolean	True if the current user has been granted any of the listed authorities.

Returns whether or not the current user has been granted any of the listed authorities.

isGrantedNone [S]

org.carewebframework.api.security.SecurityUtil

Parameter	Datatype	Description
authorities	String	A comma-delimited list of authorities to check. These may be roles and/or privileges.
<return value>	boolean	True if the current user has been granted none of the listed authorities.

Returns true if the current user has been granted none of the listed authorities.

Help Subsystem

The CWF provides support for the display of searchable and context-sensitive help content. This help subsystem provides an abstraction layer that allows it to display content from any supported format. Adapters for two of the most common formats, JavaHelp and Oracle Help for Java (OHJ), are available as CWF plugin components. The JavaHelp adapter is distributed as part of the base CWF distribution, while the OHJ adapter is packaged separately and requires the OHJ support library, available free-of-charge from the Oracle web site. An adapter for the HTML Help format is under development.

The CWF provides a help viewer that is capable of displaying specially packaged content. During the build process, a Maven plugin packages help content in its native form into a jar file that includes the addition of a Spring configuration file describing the help module. Making the help content available in a web application is a simple matter of deploying it like any other jar file. The CWF will automatically detect the help module (via the embedded Spring configuration file) and register it using its unique module id.

A plugin references a help module by specifying the name of the module in a **`help-resource`** tag in its plugin definition. When the CWF instantiates a plugin that references a help module by its unique id, it consults the Help Registry to determine if the module is available. If found, it will include that module as a selection under the help menu. For example, consider the following Spring configuration file snippet from a typical help module:

```
<beans profile="root">
   <cwh:help id="patientSelectionHelp"
      url="org/carewebframework/help/content/patientSelectionHelp/PatientSelection.hs"
      title="Patient Selection Help"
      version="3.0.0.0.89973"
      format="javahelp">
   </cwh:help>
</beans>
```

Note this uses a namespace syntax similar to that of a plugin's declaration. This configuration file is created automatically by the Maven Help Converter plugin. It defines the help module's unique id (**`patientSelectionHelp`**), the URL describing the location of the help set definition file, a user-friendly title, a content version and a content format (in this example, JavaHelp).

Now consider the plugin definition from a configuration file of a plugin that declares this help module as a resource:

```
<beans profile="root">

   <cwp:plugin
      url="~./org/regenstrief/cwf/ui/patientheader/patientHeader.zul"
      id="patientHeader"
      name="Current Patient Header">
      <cwp:resource>
         <cwp:help-resource module="patientSelectionHelp" />
      </cwp:resource>
   </cwp:plugin>

</beans>
```

Note the help resource declaration entry. This tells the CWF that whenever this plugin is instantiated, the referenced help module should be made available under the help menu.

Help Viewer

The help viewer is a ZK-based dialog that may be displayed in embedded mode (i.e., within the same browser viewport) or in a separate browser window. The viewer defaults to the latter behavior but may be changed by setting the **`org.carewebframework.help.viewer.embedded`** property to true in the **`cwf.properties`** file (see the section on CWF configuration).

HelpUtil Class

For programmatic control over the help viewer and for context-sensitive help support, the **`HelpUtil`** class provides several useful static methods:

associateCSH [S] org.carewebframework.shell.help.HelpUtil

Parameter	Datatype	Description
component	**XulElement**	**The component to be associated.**
target	**HelpTarget**	**The help target to be associated.**

Associates a ZK component with a specified help target (described below) to provide context-sensitive help. Typing F1 when the component has the focus will invoke the help viewer, displaying the specified target.

associateCSH [S] org.carewebframework.shell.help.HelpUtil

Parameter	Datatype	Description
component	XulElement	The component to be associated.
module	String	The module id of the help set.
topic	String	The topic id of the topic.
label	String	The label to be associated with the topic (may be null).

Associates a ZK component with a specified help target (given by the module and topic ids) to provide context-sensitive help. Typing F1 when the component has the focus will invoke the help viewer, displaying the specified target.

dissociateCSH [S] org.carewebframework.shell.help.HelpUtil

Parameter	Datatype	Description
component	XulElement	The component whose help target is to be dissociated.

Dissociates the specified ZK component from its help target.

getViewer [S] org.carewebframework.shell.help.HelpUtil

Parameter	Datatype	Description
<return value>	IHelpViewer	Returns a reference to the help viewer.

Returns a reference to the help viewer.

show [S] org.carewebframework.shell.help.HelpUtil

Parameter	Datatype	Description
target	HelpTarget	The help target to be displayed (see below).

Invokes the help viewer and instructs it to display the specified target.

show [S] org.carewebframework.shell.help.HelpUtil

Parameter	Datatype	Description
module	String	The module id of the help set.
topic	String	The topic id of the topic.
label	String	The label to be associated with the topic (may be null).

Invokes the help viewer and instructs it to display the specified target (as specified by module and topic ids).

showCSH [S] org.carewebframework.shell.help.HelpUtil

Parameter	Datatype	Description
component	XulElement	The component whose associated help target is to be displayed.

Invokes the help viewer and instructs it to display the help target associated with the specified ZK component. If no help target is associated with the component, the request is ignored.

showTOC [S] org.carewebframework.shell.help.HelpUtil

Invokes the help viewer and instructs it to display the table of contents.

HelpTarget Class

The **`HelpTarget`** class provides a convenient means to package help target parameters into a single object. It has the following constructor:

HelpTarget [C] org.carewebframework.shell.help.HelpTarget

Parameter	Datatype	Description
module	**String**	**The module id of the help set.**
topic	**String**	**The topic id of the topic.**
label	**String**	**The label to be associated with the topic (may be null).**

Creates a help target with the specified parameters.

Help Converter

The CWF Maven help converter plugin converts help content from a supported native format into the form required by the CWF. It is automatically invoked during the build process when it detects the following configuration properties in the **`pom.xml`** file of the project:

Property	Description
maven.carewebframework.help.moduleId	**The unique id of the help module.**
maven.carewebframework.help.moduleName	**The display name of the help module.**
maven.carewebframework.help.moduleSource	**The path of the module source relative to the `src/main/help` folder. This may be an archive file or a top level folder.**
maven.carewebframework.help.moduleFormat	**The format of the source content. Must be one of the supported formats (javahelp, ohj, etc.)**

This plugin will package the help content and the generated Spring configuration file into a jar file using the same naming convention as the main project artifact, but with a classifier name of "**`help`**" appended. To include the help content as part of a deployment, simply reference it as a Maven dependency. For example,

```
<dependencies>

   <dependency>
      <groupId>org.acme.cwf</groupId>
      <artifactId>org.acme.cwf.ui.myplugin</artifactId>
      <version>1.0.0.RELEASE</version>
   </dependency>

   <dependency>
      <groupId>org.acme.cwf</groupId>
      <artifactId>org.acme.cwf.ui.myplugin</artifactId>
      <version>1.0.0.RELEASE</version>
      <classifier>help</classifier>
   </dependency>

</dependencies>
```

Here we see a dependency for the plugin itself followed by a dependency for its help content.

Tag Libraries

The CWF distribution includes two tag libraries that provide convenient access to commonly used framework services. You may include references to one or both of these tag libraries in your zul page as shown in this example:

```
<?taglib uri="http://www.carewebframework.org/tld/core" prefix="rc"?>
<?taglib uri="http://www.carewebframework.org/tld/security" prefix="sec"?>

<zk>
   <div visible="${sec:isAuthenticated}>"
      <image src="${rc:getIconPath('information.png')}" />
   </div>
</zk>
```

Core Tag Library

The core tag library exposes methods for referencing web resources such as icons. It supports the following methods:

getCookie http://www.carewebframework.org/tld/core

Parameter	Datatype	Description
cookieName	**String**	**The name of a cookie.**
<return value>	**String**	**A cookie value, or null if not found.**

Returns the fully decoded value of the named cookie from the current request.

getFellow http://www.carewebframework.org/tld/core

Parameter	Datatype	Description
id	**String**	**A ZK component id.**
<return value>	**Component**	**The component with a matching id. Throws an exception if none found.**

Returns a ZK component from the current ID space that has the specified identifier.

getFellowIfAny http://www.carewebframework.org/tld/core

Parameter	Datatype	Description
id	**String**	**A ZK component id.**
<return value>	**String**	**The component with a matching id. Returns null if none found.**

Returns a ZK component from the current ID space that has the specified identifier.

getIconPath http://www.carewebframework.org/tld/core

Parameter	Datatype	Description
iconName	String	The name of the icon resource.
<return value>	String	The URL of the requested resource, or null if none found.

Returns the URL of an icon resource from the default icon library using the default dimensions. See the section on icon libraries for more information.

getIconPathEx http://www.carewebframework.org/tld/core

Parameter	Datatype	Description
iconName	String	The name of the icon resource.
dimensions	String	The desired pixel dimensions (e.g., 16x16).
library	String	The name of the icon library.
<return value>	String	The URL of the requested resource, or null if none found.

Returns the URL of an icon resource of the specified dimensions from the specified icon library. See the section on icon libraries for more information.

Security Tag Library

This tag library provides convenient access to security-related functions. It can be used, for example, to conditionally expose parts of the UI depending on a user's granted security authorities.

isAuthenticated http://www.carewebframework.org/tld/security

Parameter	Datatype	Description
<return value>	boolean	True if the current execution context has been authenticated.

Returns whether or not the current execution context has been authenticated.

isGrantedAll http://www.carewebframework.org/tld/security

Parameter	Datatype	Description
authorities	String	A comma-delimited list of authorities to be checked. These may be security privileges or roles.
<return value>	boolean	True if the current user has been granted all of the listed authorities.

Returns true if the current user has been granted all of the listed authorities.

isGrantedAny

http://www.carewebframework.org/tld/security

Parameter	Datatype	Description
authorities	String	A comma-delimited list of authorities to be checked. These may be security privileges or roles.
<return value>	boolean	True if the current user has been granted any of the listed authorities.

Returns true if the current user has been granted any one of the listed authorities.

isGrantedNone

http://www.carewebframework.org/tld/security

Parameter	Datatype	Description
authorities	String	A comma-delimited list of authorities to be checked. These may be security privileges or roles.
<return value>	boolean	True if the current user has been granted none of the listed authorities.

Returns true if the current user has been granted none of the listed authorities.

getUsername

http://www.carewebframework.org/tld/security

Parameter	Datatype	Description
<return value>	String	The username of the currently authenticated user, or null if authentication has not yet occurred.

Returns the username of the currently authenticated user.

Printing Support

Printing from a web-based application is often challenging. There is no single solution that meets all needs. The CWF provides a convenient JavaScript solution that is adequate for many simple printing needs. It supports printing directly to a client's printer as well as displaying a print preview that can subsequently be printed as a separate step. It assumes that you wish to print elements restricted to a specific portion of the browser's document. Consider the following zul page fragment:

```
<panel>
  <toolbar>
    <button id="btnPrint" label="Print"
      w:onClick="cwf.print(this.$f('printRoot'),'~./clinicalDocumentsPrint.css');" />
  </toolbar>
  <panelchildren>
    <div>
      <listbox id="lstDocuments" />
    </div>
    <div id="printRoot">
      <listbox id="lstDisplay" />
    </div>
  </panelchildren>
</panel>
```

This page has a button labeled "Print" with an associated client-side action that invokes some JavaScript code when it is clicked. This code calls a JavaScript function called **`cwf.print`**, which has the following signature:

cwf.print org/carewebframework/ui/zk/common.js

Parameter	Datatype	Description
source	**DOM Element or ZK Widget**	**The DOM element or ZK widget (or array of same) that provides the content for the printed output.**
printStyles (optional)	**String or String[]**	**A comma-delimited string or an array of style sheet references that will be applied to the printed output.**
printPreview (optional)	**boolean**	**If true, display as a print preview in a separate browser window. If false (the default), print directly to a user-selected printer.**

This function concatenates the portion of the DOM document rooted at each of the specified source elements, applies any referenced style sheets, and either sends it to a user-selected printer or displays it as a print preview in a separate browser window.

Referring again to the zul page example above, clicking the print button would invoke the **`cwf.print`** function, passing it a reference to the ZK widget with an id of **`printRoot`** (the **`$f`** JavaScript function is a ZK-defined function that returns the fellow widget named in its single parameter). It instructs the function to apply a style sheet before printing. Only that portion of the document rooted at the **`printRoot`** ZK widget (i.e., the list box with an id of **`lstDisplay`**) will be printed.

There are times when you may have a generic style sheet that you always want applied before printing. It is cumbersome to specify this style sheet reference every time you call **`cwf.print`**. In this case, you may want to register your style sheet to be automatically applied with each print request. To do this, you may use the JavaScript function **`cwf.registerPrintStyle`**:

cwf.registerPrintStyle org/carewebframework/ui/zk/common.js

Parameter	Datatype	Description
printStyle	**String**	**The URL of the style sheet to be applied for every print operation.**

A convenient location to place this code is inside a **`javascript`** tag in the **`lang-addon.xml`** file for your plugin (see the ZK documentation for further information on this configuration file). For example,

```
<language-addon>
   <addon-name>org.regenstrief.cwf.ui.resultsdisplay.addon</addon-name>
   <depends>cwf.ui</depends>
   <language-name>xul/html</language-name>

   <javascript-module name="org.acme.cwf.ui.resultsdisplay.addon" version="2.0.0"/>

   <javascript>
      cwf.registerPrintStyle('~./org/acme/cwf/ui/resultsdisplay/resultsDisplayPrint.css');
   </javascript>

   <stylesheet href="~./org/acme/cwf/ui/resultsdisplay/resultsDisplay.css" type="text/css" />

</language-addon>
```

The above configuration would cause the referenced style sheet to be applied for every print operation. Note that style sheets registered via the **`cwf.registerPrintStyle`** call are applied before style sheets referenced in the **`cwf.print`** function, allowing the latter to override settings from the former.

The JavaScript printing functions may also be invoked from server-side code. The methods supporting this capability are described in the section on the **`ZKUtil`** class and are repeated here for convenient reference:

printToClient [S] org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
selectors	**List <String>**	**A list of DOM selectors whose content will be printed.**
styleSheets	**List <String>**	**A list of style sheet refererences to be applied to the printed output.**
preview	**boolean**	**If true, open in preview mode. If false, submit directly for printing.**

Prints portions of the DOM to the client via the browser's native printing capability. See the section on printing for more detail.

printToClient [S] org.carewebframework.ui.zk.ZKUtil

Parameter	Datatype	Description
selector	**String**	**A DOM selector whose content will be printed.**
styleSheet	**String**	**The url of a style sheet to be applied to printed output. May be null.**
preview	**boolean**	**If true, open in preview mode. If false, submit directly for printing.**

Prints a portion of the DOM to the client via the browser's native printing capability. See the section on printing for more detail.

Theme Support

The CWF provides support for alternative styling of the UI through theme plugins. A theme plugin is a jar file that contains replacement style sheets and images that may be substituted on the fly as they are requested by the browser. These plugins are built using a Maven plugin, the ZK Theme Generator. Given a base color, the ZK Theme Generator plugin will process selected jar files (these will typically be the jar files in the ZK distribution as well as any other jar files containing resources to be re-themed), extract any embedded style sheets and image files, apply a color warping algorithm that will transform colors relative to the new base color, and package these modified resources into a jar file along with a Spring configuration file entry that contains a descriptor for the theme.

Each plugin theme is automatically discovered by the CWF at application startup and registered with the Theme Registry. A theme may be selected by referencing it by name in a query parameter appended to the application URL (e.g., **`https://…?theme=theme-green`**) or by setting the **`CAREWEB.THEME`** user preference to the theme name.

Note: The next version of the ZK Framework, version 7.0, will integrate support for Twitter's Bootstrap theme engine and the LESS style sheet preprocessor, which will not only simplify the creation of alternative themes, but also provide access to a large repository of ready-made themes. We anticipate significant changes to our theme support once we incorporate this new version.

Icon Libraries

An icon library is a collection of icon images wrapped by a special API that allows them to be registered and referenced in a consistent manner. Multiple icon libraries may be registered at the same time. The CWF property editor can display the contents of registered icon libraries when editing icon type properties.

Two configuration properties are relevant to API behavior:

Property Name	Default Value	Description
org.carewebframework.icons.library.default	<none>	The default icon library.
org.carewebframework.icons.dimensions.default	16x16	The default icon dimensions.

The CWF distribution supplies an icon library based on the Silk icon collection from Mark James (www.famfamfam.com).

IIconLibrary Interface

The **`IIconLibrary`** interface provides a consistent mechanism to access image resources from an image collection. It declares the following methods:

getIconUrl — org.carewebframework.ui.icons.IIconLibrary

Parameter	Datatype	Description
iconName	String	The icon's file name.
dimensions	String	The desired dimensions (e.g., "16x16"). May be null.
<return value>	String	The URL of the icon resource. Will be null if no matching resource was found.

Returns the URL of the requested icon resource, or null if none found.

getId — org.carewebframework.ui.icons.IIconLibrary

Parameter	Datatype	Description
<return value>	String	The library's unique identifier (e.g., "silk").

Returns the unique identifier of the library.

getMatching — org.carewebframework.ui.icons.IIconLibrary

Parameter	Datatype	Description
iconName	String	The icon's file name (with or without wildcards). May be null.
dimensions	String	The desired dimensions (with or without wildcards). May be null.
<return value>	List <String>	A list of URLs for icon resources that match the specified criteria.

Returns a list of URLs for icon resources that match the specified criteria. Both the **`iconName`** and the **`dimensions`** parameters may contain wildcard characters, or may be null to match all resources. For example,

```
getMatching("weather*", "*x16")
```

or

```
getMatching(null, "16x16")
```

supportedDimensions org.carewebframework.ui.icons.IIconLibrary

Parameter	Datatype	Description
<return value>	String[]	An array of pixel dimensions available from this library.

Returns an array of pixel dimensions that are available from this library.

IconLibraryBase Class

The **`IconLibraryBase`** class is a reference implementation of the **`IIconLibrary`** interface. It should be suitable for implementing most icon libraries provided they follow a subscribed pattern for organizing icon resources (see below). As expected, the class implements all of the methods of the **`IIconLibrary`** interface as documented above. The single constructor has the following signature:

IconLibraryBase [C] org.carewebframework.ui.icons.IconLibraryBase

Parameter	Datatype	Description
id	String	The unique id of the library (e.g., "silk").
resourcePath	String	The root path of the icon resources.
dimensions	String	The dimensions supported by the library, in pixels. Separate multiple entries with commas (e.g., "16x16,32x32").

Creates an instance of the icon library base class with the specified parameters.

IconUtil Class

This utility class provides static convenience methods for interacting with the icon library registry:

getIconPath [S] org.carewebframework.ui.icons.IconUtil

Parameter	Datatype	Description
iconName	String	The icon's file name.
dimensions (optional)	String	The desired dimensions. If not specified, the default dimensions are used.
library (optional)	String	The unique id of the icon library. If not specified, the default library is used.
<return value>	String	The URL of the requested resource.

Returns the URL of the requested icon, or null if none found.

getMatching [S] org.carewebframework.ui.icons.IconUtil

Parameter	Datatype	Description
iconName	String	The icon's file name (with or without wildcards). Specify null to match any file name.
dimensions	String	The desired dimensions (with or without wildcards). Specify null to match any dimension.
library	String	The unique id of the icon library (with or without wildcards). Specify null to use the default icon library.
<return value>	List <String>	A list of URLs for icon resources that match the specified criteria.

Returns a list of URLs for icon resources that match the specified criteria. All three parameters may contain wildcard characters, or may be null. For example,

```
IconUtil.getMatching("silk*", "weather*", "*x16")
```

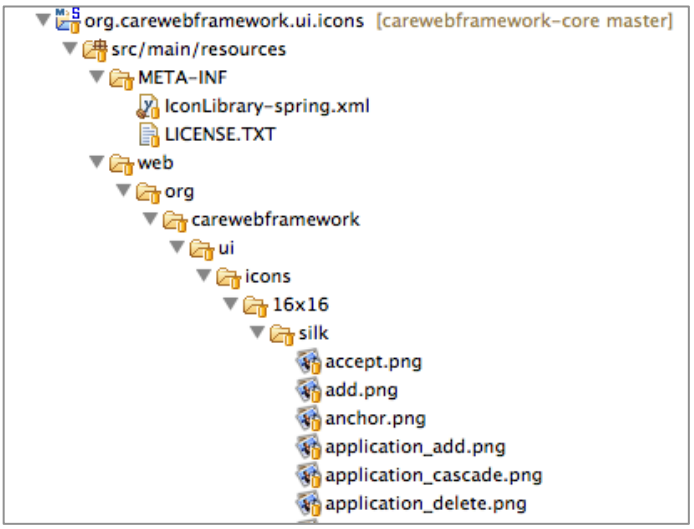
or

```
IconUtil.getMatching(null, null, "16x16")
```

Creating an Icon Library

The Silk icon library (Maven artifact id **`org.carewebframework.ui.icons`**) can serve as a guide for creating an icon library. If you use the same folder organization for the icon image resources, no coding is necessary. Otherwise, you will need to create your own **`IIconLibrary`** implementation.

Looking at the Silk icon library project contents, one can see that the project is very straightforward:



The project consists of a Spring configuration file and the icon images themselves. Note the folder path where the icon images are located. The root of the path is, as expected, the **`web`** folder. This is followed by the familiar folder naming convention

that mirrors the package (or Maven artifact) name, in this case **`org/carewebframework/ui/icons`**. This portion of the path constitutes the root path of the icon library. Next, we see a folder whose name represents the pixel dimensions of the icon images, **`16x16`**. Finally, the bottom level folder is the library name itself, **`silk`**. This is the folder organization expected by the **`IconLibraryBase`** class.

Now let's look at the Spring configuration file from the project, **`IconLibrary-spring.xml`**:

```
<beans profile="root">

   <bean class="org.carewebframework.ui.icons.IconLibraryBase">
      <constructor-arg value="silk" />
      <constructor-arg value="/org/carewebframework/ui/icons/" />
      <constructor-arg value="16x16" />
   </bean>

</beans>
```

This file contains a single bean definition that simply creates an instance of the **`IconLibraryBase`** class with three constructor arguments: the library name, the root path, and the image dimensions. If the Silk library had more than one image size, we could have included these in the third constructor argument, separating them with commas.

That's all there is to creating an icon library. Because **`IconLibraryBase`** implements the **`IIconLibrary`** interface, it will automatically be registered with the icon library registry.

Configuration

Properties

Properties come in several flavors: layout properties, Maven properties, configuration properties and domain properties. All control various behaviors of the CWF and components running within it.

Layout Properties

Layout properties have been discussed elsewhere. These are properties associated with layout components that are persisted as part of the layout snapshot. They may be modified using the property grid while in design mode.

Maven Properties

Maven properties are specified in the **`properties`** section of the **`pom.xml`** file and affect the build process. We have included information about specific Maven properties in the respective section where they are relevant. A detailed treatment of Maven properties is beyond the scope of this document.

Configuration Properties

Configuration properties are stored in property files on the local file system of the running application. Generally speaking, configuration properties are used to control initial startup of the CWF and its core services. Configuration properties are relatively static in nature and are global in their scope. A configuration property value is typically injected into its target bean by the Spring IOC container using an EL style reference. For example:

```
<bean id="taskScheduler"
   class="org.springframework.scheduling.concurrent.ScheduledExecutorFactoryBean"
   init-method="initialize" destroy-method="destroy">

   <property name="threadNamePrefix"
      value="${task.scheduler.thread.factory.threadnameprefix}" />

   <property name="poolSize"
      value="${task.scheduler.pool.size}" />

   <property name="waitForTasksToCompleteOnShutdown"
      value="${task.scheduler.waitfortaskstocompleteonshutdown}" />

   <property name="awaitTerminationSeconds"
      value="${task.scheduler.awaitterminationseconds}" />

</bean>
```

The **cwf.properties** file contains property settings that affect various behaviors of the CWF and components running within it. This file must be located on the application class path. In a web application, it is typically placed in the **WEB-INF/classes** folder. The property values specified here can be a combination of framework settings and domain-specific settings and will override any default settings specified elsewhere. A sample property file is shown below:

```
# How often to update the displayed count down timer (in ms).
org.carewebframework.ui.DesktopMonitor.COUNTDOWN_INTERVAL=2000

# Enable embedded mode for help viewer.
org.carewebframework.help.viewer.embedded=true

# JDBC URL for connecting to the database server
jdbc.url=jdbc:oracle:thin:@192.168.2.1:1521:dbserver
# JDBC Database driver classname to load
jdbc.driverClassName=oracle.jdbc.OracleDriver
# Database adapter classname for implementation-specific APIs.
jdbc.adapterClassName=org.regenstrief.database.oracle.OracleAdapter
# JDBC username and password for logging into the database server
jdbc.username=dbuser
jdbc.password=xxxxxxx


# User login for test harness
test.username=testuser
test.password=xxxxxxx
test.domain=WISHARD MEMORIAL HOSPITAL
```

The following is an alphabetical list of CWF configuration properties and their function:

Property Name	Default Value	Description
org.carewebframework.api.aliases	<none>	Location of a property file containing allas assignments.
org.carewebframework.help.viewer.embedded	false	If true, the help viewer displays in the current viewport. If false, it displays in a separate browser window.
org.carewebframework.icons.dimensions.default	16x16	The default icon dimensions.
org.carewebframework.icons.library.default	<none>	The default icon library.
org.carewebframework.jms.activemq.broker.name	localhost-org.carewebframework	Should be unique across broker network.
org.carewebframework.jms.activemq.broker.network.uri	<default>	Connection URI for network broker.
org.carewebframework.jms.activemq.broker.persistence.dir	<tempdir>/jms/data	Location of persistence data store.
org.carewebframework.jms.activemq.broker.persistence.maxsize	32mb	Maximum storage capacity for local persistence data store.
org.carewebframework.jms.activemq.broker.transport.connector.url	tcp://localhost:0	Additional non-VM transport.
org.carewebframework.jms.activemq.broker.url	<default>	Connection URL for local broker.
org.carewebframework.jms.activemq.connection.client.id	<same as local name>	Client id.
org.carewebframework.jms.connection.factory.session.cache.size	1	Session cache size.
org.carewebframework.property.location	classpath:cwf*.properties	The location of property files.
org.carewebframework.security.login.url	<default login dialog>	The redirection target after successful login.
org.carewebframework.security.logout.url	<default logout dialog>	The redirection target after logging out.
org.carewebframework.thread.executor.core.pool.size	10	The executor's core pool size.
org.carewebframework.thread.executor.max.pool.size	10	The executor's maximum pool size.
org.carewebframework.thread.executor.queue.capacity	50	The capacity for the blocking queue.
org.carewebframework.thread.executor.shutdown.wait	true	If true, the executor will wait for all scheduled tasks to complete on shutdown.
org.carewebframework.thread.executor.shutdown.timeout	10	The maximum # of seconds to wait for remaining tasks to complete.
org.carewebframework.thread.executor.thread.name.prefix	taskExecutor-	The prefix for names of newly created threads.
org.carewebframework.thread.scheduler.pool.size	10	The scheduler's pool size.
org.carewebframework.thread.scheduler.shutdown.wait	true	If true, the scheduler will wait for all scheduled tasks to complete on shutdown.
org.carewebframework.thread.scheduler.shutdown.timeout	10	The maximum # of seconds to wait for remaining tasks to complete.

org.carewebframework.thread.scheduler.thread.name.prefix	**taskScheduler-**	**The prefix for names of newly created threads.**
org.carewebframework.ui.command.shortcuts	**classpath:shortcut.properties**	**Location of property file containing shortcut assignments.**
org.carewebframework.ui.desktop.countdown.duration.baseline	**60000**	**Length of countdown (ms) in baseline state.**
org.carewebframework.ui.desktop.countdown.duration.lock	**60000**	**Length of countdown (ms) in lock state.**
org.carewebframework.ui.desktop.countdown.interval	**2000**	**How often (ms) to update the displayed countdown timer.**
org.carewebframework.ui.desktop.inactivity.duration.baseline	**900000**	**Length of inactivity (ms) in baseline state before presenting timeout dialog.**
org.carewebframework.ui.desktop.inactivity.duration.lock	**900000**	**Length of inactivity (ms) in locked state before presenting timeout dialog.**
org.carewebframework.ui.desktop.inactivity.interval.maximum	**300000**	**Maximum interval of inactivity (ms) after which a desktop is assumed to be dead.**
org.carewebframework.ui.desktop.lock.exclusions	**<none>**	**Comma-delimited list of application names to exclude from locking.**

Domain Properties

Domain properties are stored on the host system in a domain-dependent fashion. Domain properties are typically accessed via the Property Service, which provides a level of abstraction between property access and property storage. While the CWF provides a default implementation of the Property Service, most deployments will implement a custom version that uses the data store of the underlying host system. Domain properties can be scoped globally or by user.

The following is an alphabetical list of CWF domain properties and their function:

Property Name	Default Value	Description
CAREWEB.APPID.DEFAULT	-	The default application id.
CAREWEB.CONFIRM.CLOSE	true	If true, the user is prompted before closing the browser viewport.
CAREWEB.INITIAL.SECTION	-	Designates which plugin is initially active when a layout is loaded.
CAREWEB.LAYOUT.ASSOCIATION	-	Stores the association between an application id and its default layout.
CAREWEB.LAYOUT.DEFAULT	-	Stores the name of the default layout.
CAREWEB.LAYOUT.PRIVATE	-	Stores private layouts.
CAREWEB.LAYOUT.SHARED	-	Stores shared layouts.
CAREWEB.SAVED.RESPONSES	-	Stores default dialog responses.
CAREWEB.THEME	-	The default theme to use.

Web Application

The **`web.xml`** and **`zk.xml`** files found in the **`WEB-INF`** folder of the web application control various behaviors of the ZK and Spring Frameworks. The CWF requires several additional entries to the default versions of these files. The versions found in the test harness web app are properly configured for the CWF and should be used as a starting point for any CWF web deployment.

References

Stäble, M., H.-J. Schumacher, et al. (2008). "ZK developer's guide developing responsive user interfaces for web applications using AJAX, XUL, and the open-source ZK rich web client development framework." Packt Publishing.

Chen, H., R. Cheng, et al. (2007). "ZK Ajax without JavaScript Framework." firstPress/Apress.

Johnson, R. and Books24x7 Inc. (2005). "Professional Java development with the Spring Framework." Wiley Pub.

Van de Velde, T., A. Horton, et al. (2008). "Beginning Spring Framework 2." Wiley Pub.

Health Level Seven (2000). "Context Management Specification Version CM-1.3."