Inside WebObjects

WebObjects Dynamic Elements Reference

February 2002

Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada

Contents

Dynamic Element Specifications

Dynamic elements serve as the basic building blocks of WebObjects applications by linking an application's scripted or compiled behavior to an HTML page. The linkage can be two-way, in that a dynamic element:

- Initially sets its attributes to values specified by scripted or compiled variables or methods.
- Represents itself as HTML when called upon to do so.
- Synchronizes the values of its attributes to those entered by the user, and passes these values back to your script or compiled code.

With WebObjects, most pages sent to the user's browser are composed of HTML from a static template combined with HTML that's dynamically generated by dynamic elements embedded (directly, or in the case of reusable components, indirectly) in that template.

Here are the dynamic elements defined in the WebObjects Framework:

```
WOActionURL
WOActiveImage
WOApplet
WOBody
WOBrowser
WOCheckBox
WOCheckBoxList
WOComponentContent
WOConditional
WOEmbeddedObject
WOFileUpload
```

WOForm
WOFrame
WOGenericContainer
WOGenericElement
WOHiddenField
WOHyperlink
WOImage
WOImageButton
WOJavaScript
WONestedList
WOParam
WOPasswordField
WOPopUpButton
WOQuickTime
WORadioButton
WORadioButtonList
WORepitition
WOResetButton
WOResourceURL
WOString
WOSubmitButton
WOSwitchComponent
WOText
WOTextField
WOVBScript

How to Use These Specifications

Each dynamic element specification that follows is divided into three sections: a synopsis, a description, and a set of bindings. The synopsis is designed to give you ready reference to the element's attributes, showing which ones are mandatory and which ones optional. The description explains the purpose of the element. Finally, the bindings describe in detail each of the dynamic element's attributes.

The element synopses use several conventions that you should be aware of, for example:

WOVBScript { scriptFile=*aPath* | scriptString=*aString* | scriptSource=*aURL*; [hideInComment=*aBoolean*;] ... };

- *Italic* denotes words that represent something else or that can be varied. For example, *aPath* represents a path to a file—the exact path is your choice.
- Square brackets ([ ]) mean that the enclosed attribute or attributes are optional. The name attribute and its value are optional in the synopsis above.
- A vertical bar (|) separates options that are mutually exclusive, as in scriptFile=*aPath* | scriptString=*aString* | scriptSource=*aURL*, where you can specify either a `scriptFile`, a `scriptString`, or a `scriptSource`, but not some combination of the three.
- Ellipsis (...) represents additional key-value pairs that you might add but that aren't part of the element's specification. When a dynamic element is asked to produce its HTML representation, these additional key-value pairs are simply copied inside the opening tag of the HTML element the dynamic element corresponds to, after all the dynamically generated attributes and keys. Abstract dynamic elements that do not correspond to an HTML tag can not display any additional attributes. You can use the `otherTagString` attribute to specify optional attributes that are not key-value pairs.
- The remaining words or characters are to be taken literally (that is, they should be typed as they appear). For example, the `scriptFile` and `scriptString` and other attribute names are to be take literally in the synopsis above.

The `otherTagString` Attribute

All dynamic elements include an optional attribute, `otherTagString`, Use this attribute to have the bound string included directly in an element's HTML tag. Some HTML elements contain parameters that are not key-value pairs. If you wish to include one of these parameters in your element, you can send it using this attribute.

WOActionURL

WOActionURL enables the creation of URLs to invoke methods or specify pages to return. You can use this element for a variety of purposes, but it is primarily intended to support JavaScript within a WebObjects application.

Synopsis

WOActionURL {action=*aMethod* | pageName=*aString*;
directActionName=*anActionName*; actionClass=*className*;
[fragmentIdentifier=*anchorFragment*;] [queryDictionary=*aDict*; ?key=*value*;]
[secure=*aBoolean*;]}

Bindings

action

Action method to invoke when the URL is accessed. This method must return a an object that conforms to the `WOActionResults` interface such as `WOComponent` or `WOResponse`.

pageName

The name of a WebObjects page to display when the URL is accessed.

WOActionURL

directActionName

The direct action method to invoke when the URL is accessed (minus the "Action" suffix). Defaults to “default”.

actionClass

The name of the class in which the `directActionName` can be found. Defaults to “DirectAction”.

fragmentIdentifier

Named location to display in the destination page (that is, an anchor in the destination page).

queryDictionary

NSDictionary with key-value pairs to be placed into the URLs query string.

?key

Adds a key-value pair to the specified `queryDictionary` (or replaces an existing key) by prefixing the key with a "?". For example: `?x = y;` puts the key "x" into the query dictionary with the value of the keypath y.

secure

Changes the URL prefix from `http` to `https` when WebObjects generates URLs for component actions and direct actions for this element. For this attribute to have any effect, you must provide bindings either for the *action*, *directAction*, *actionClass*, or *pageName* attribute (respecting the valid combinations).

WOActiveImage

A WOActiveImage displays an image within the HTML page. If the WOActiveImage is disabled, it simply displays its image as a passive element in the page. If enabled, the image is active; that is, clicking the image generates a request.

WOActiveImages are intended to be used outside of an HTML form. WOActiveImage functions as a mapped, active image. When the user clicks in a WOActiveImage, the coordinates of the click are sent back to the server. Depending on where the user clicks, different actions are invoked. An image map file associates actions with each of the defined areas of the image. If an image map file is not specified, the method specified by the action attribute is performed when the image is clicked, or if the `href` attribute is specified, the image acts as a hyperlink and takes you to that destination.

Within an HTML form, a WOActiveImage functions as a graphical submit button. However, it is better to use a WOImageButton instead of WOActiveImage to create a graphic submit button or a mapped image within a form.

Synopsis

WOActiveImage {filename= *imageFileName*;
[framework = *frameworkBaseName*|"app";] | src=*aURL*; | value=*aMethod*;
action=*aMethod* | href=*aURL* | actionClass=*aClass* | directActionName=*aName*; |
data=*dataObject*; mimeType=*typeString*; [key=*cacheKey*;] [imageMapFile=*aString*];
[name=*aString*;] [x=*aNumber*; y=*aNumber*;] [target=*frameName*;] [disabled=*aBoolean*;]
[secure=*aBoolean*;]...}

Bindings

filename

Path to the image relative to the WebServerResources directory.

framework

Framework that contains the image file. This attribute is only necessary if the image file is in a different location from the component. That is, if the component and the image file are both in the application or if the component and the image file are both in the same framework, this attribute isn't necessary. If the image file is in a framework and the component is in the application, specify the framework's name here (minus the .framework extension). If the image file should be in the application but the component is in a framework, specify the "app" keyword in place of the framework name.

src

URL containing the image data. Use this attribute for complete URLs; for relative URLs use `filename` instead.

value

Image data in the form of a `WOElement` object. This data can come from a database, a file, or memory.

action

Method to invoke when this element is clicked. If `imageMapFile` is specified, `action` is only invoked if the click is outside any mapped area. In other words, `action` defines the default action of the active image.

href

URL to direct the browser to as a default when the image is clicked and no hot zones are hit.

actionClass

The name of the class in which the `directActionName` can be found. Defaults to "DirectAction".

directActionName

The direct action method to invoke when the URL is accessed (minus the "Action" suffix). Defaults to "default".

data

Specifies an image resource in the form of an `NSData`; this data can come from a database, a file, or memory. If you specify resource data, you must specify a MIME type.

mimeType

A string designating a MIME resource type, such as "image/gif", to be put in the content-type header field; this type tells the client what to do with data. If you provide `data` but no MIME type, an exception is thrown.

key

A string that the application uses as a key for caching the data specified in `data`. If you do not provide a key, the data object must be fetched each time it is needed. For further information, see the reference documentation for the `WOResourceManager` class, (in particular, see the `flushDataCache` method).

imageMapFile

Name of the image map file. See "The Image Map File" (page 18) for more details.

name

If `name` is specified then the hit point is specified as name.x=*value*; name.y=*value*; in the form. This is useful when you need to use this element to submit a form to an external URL that expects the hit point to be expressed in a certain format.

x, y

If specified, returns the coordinates of the user's click within the image.

target

Frame in a frameset that will receive the page returned as a result of the user's click.

disabled

If disabled evaluates to `true`, a regular image element (`<IMG>`) is generated rather than an active image.

secure

Changes the URL prefix from `http` to `https` when WebObjects generates URLs for component actions and direct actions for this element. For this attribute to have any effect, you must provide bindings either for the *action*, *directAction*, *actionClass*, or *pageName* attribute (respecting the valid combinations).

The Image Map File

If the `imageMapFile` is specified, WebObjects searches for the image map file in the application and, if the image is in a framework, the search continues in the framework where the image resides. You should put image map files into the Resources group in your Project Builder project. If your project has localized images, the image map file may also need to be localized. If you choose to have localized mapped images, you must have a corresponding map file for each localized image (unless you only have one map file which is not in any locale-specific .lproj directory).

The `imageMapFile` binding must be in quotation marks and the image map file must be a plain text file. So, if the image map file is "map.txt," you must add it to your project (to the Resources group and Application Server target) and specify the binding for `imageMapFile` as the literal string "map.txt".

Note: The image map file must be in the same location as the image. For example, if the image is in a framework, the image map file must be in that same framework.

Each line in the image map file has this format:

shape action coordinate-list

shape

Either `rect`, `circle`, or `poly`. For a `rect` shape, the coordinates x1,y1 specify the upper-left corner of the hot zone, and x2,y2 specify lower-right corner. For a `circle` shape, the x1,y1 is the origin, and x2,y2 is a point on the circle. For the `poly` shape, each coordinate is a vertex: up to 100 vertices are supported.

action

Name of the component action method to invoke when the image is clicked. To specify a direct action, provide a fully-qualified method name such as "com.mycompany.MyDirectActionClass.aDirectActionMethod".

coordinate-list

The list of coordinates (x1,y1 x2,y2 ...) as described under *shape*, above.

Below are some sample entries in an image map file. Be careful to include a space character between each coordinate but to not include a space character after commas.

These samples specify component actions (the methods “home” and “buy” in the component class):

```
rect home 0,0 135,56
rect buy 135,0 270,56
```

This sample specifies a direct action method “foo” in the DirectAction class:

```
rect DirectAction.foo 0,0 256,256
```

This sample specifies a direct action method “foo” in a class called “MyDirectActionClass” in the package “com.mycompany:”

```
rect com.mycompany.MyDirectActionClass.foo 0,0 256,256
```

WOApplet

WOApplet is a dynamic element that generates HTML to specify a Java applet. The applet's parameters are passed by one or more WOParam elements.

Synopsis

WOApplet { code=*javaClassName*; width=*aWidth*; height=*aHeight*; [associationClass=*className*;] [codeBase=*aPath*;] [archive=*jarFile1*[, *jarFile2*];] [archiveNames=*jarFile1*[, *jarFile2*];] [object=*serializedApplet*;] [hspace= *aSize*;] [vspace=*aSize*;] [align=*aString*]... }

Bindings

code
: Name of the Java class.

width
: Width, in pixels, of the area to allocate for the applet.

height
: Height, in pixels, of the area to allocate for the applet.

associationClass
: Name of the Java subclass of WOAssociation that aids in communication between client applet and the server.

WOApplet

codeBase

Directory that contains the applet code. If this attribute is omitted, the applet code is assumed to be in the same directory as the template HTML file.

archive

Comma-separated list of URLs for jar archive files containing classes and other resources that will be preloaded. (Note: Currently, most browsers do not support a comma-separated list, so only a single archive file may be used.) Use this attribute for archive files that you have generated outside of a WebObjects application or framework. The value for this attribute is appended to the `archiveNames` attribute value.

archiveNames

Comma-separated list of archive files containing classes and other resources that will be preloaded. (Note: Currently, most browsers do not support a comma-separated list, so only a single archive file may be used.) Use this attribute for archive files that are built as part of a WebObjects application or framework project.

object

File containing serialized representation of the applet.

hspace

Amount of whitespace (in pixels) to the left and right of the applet.

vspace

Amount of whitespace (in pixels) at the top and bottom of the applet.

align

Alignment of the applet. Possible values are top, bottom, left, right, and middle.

WOBody

WOBody specifies the background image to display for the HTML page. All bindings for this element are related to the background image.

Synopsis

WOBody {src=*aURL* | filename= *imageFileName*; [framework = *frameworkBaseName* | "app" ;] | data=*dataObject*; mimeType=*typeString*; [key=*cacheKey*;]... }

Bindings

src

URL containing the image data. Use this attribute for complete URLs; for relative URLs use `filename` instead.

filename

Path to the image relative to the WebServerResources directory.

framework

Framework that contains the image file. This attribute is only necessary if the image file is in a different location from the component. That is, if the component and the image file are both in the application or if the component and the image file are both in the same framework, this

attribute isn't necessary. If the image file is in a framework and the component is in an application, specify the framework's name here (minus the `.framework` extension). If the image file should be in the application but the component is in a framework, specify the "app" keyword in place of the framework name.

data

Specifies any resource in the form of an `NSData` object; this data can come from a database, a file, or memory. If you specify resource data, you must specify a MIME type.

mimeType

A string designating a MIME resource type, such as "image/gif"; this type tells the client what to do with data. If you provide `data` but no MIME type, WebObjects throws an exception.

key

A string that functions as a key for caching the data specified in `data`. If you do not provide a key, the data object must be fetched each time it is needed. For further information, see the reference documentation for the `WOResourceManager` class (pay particular attention to the `flushDataCache` method).

WOBrowser

WOBrowser displays itself as a selection list that displays multiple items at a time. The related element WOPopUpButton is similar to WOBrowser except that it restricts the display to only one item at a time.

You should provide the title of an item in `displayString` rather than in `value`. If there is no binding for `displayString`, the string assigned to `value` is used for the item.

Synopsis

WOBrowser { list=*anArray*; item=*anItem*; [displayString=*displayValue*; value=*optionValue*;] [escapeHTML=*aBoolean*;] [selections=*objectArray*; | selectedValues=*valueArray*;] [name=*fieldName*;] [disabled=*aBoolean*;] [multiple = *aBoolean*;] [size=*anInt*;]... }

Bindings

list

Array of objects from which the browser derives its values. For example, colleges could name the list containing objects that represent individual schools.

item

Identifier for the elements of the list. For example, `aCollege` could represent an object in the colleges array.

displayString

Value to display in the selection list; for example, `aCollege.name` for each college object in the list.

value

For each OPTION tag within the selection, this is the `value` attribute (that is, <OPTION value=*someValue*>). This value can be used as an identifier of an item in the list.

escapeHTML

If `escapeHTML` evaluates to `true`, the string rendered by `displayString` is converted so that characters which would be interpreted as HTML control characters become their escaped equivalent (this is the default). Thus, if a your `displayString` is "`a <b>bold</b> idea`", the string passed to the client browser would be "`a <B>bold</B> idea`", but it would display in the browser as "`a <b>bold</b> idea`". If `escapeHTML` evaluates to `false`, WebObjects simply passes your data to the client browser "as is." In this case, the above example would display in the client browser as "a **bold** idea". If you are certain that your strings have no characters in them which might be interpreted as HTML control characters, you get better performance if you set `escapeHTML` to `false`.

selections

Array of objects that the user chose from list. For the college example, selections would hold college objects.

selectedValues

Array of values that is used with Direct Actions to specify which options in a list are selected.

name

Name that uniquely identifies this element within the form. You can specify a name or let WebObjects automatically assign one at runtime.

disabled

If `disabled` evaluates to `true`, this element appears in the page but is not active. That is, selections won't contain the user's selection when the page is submitted.

multiple

If `multiple` evaluates to `true`, the user can select multiple items from the list. Otherwise, the user can select only one item from the list. The default is `false`.

size

How many items to display at one time. The default is 5. `size` must be greater than 1.

CHAPTER 6

WOBrowser

WOCheckBox

A WOCheckBox object displays itself in the HTML page as its namesake, a check box user interface control. It corresponds to the HTML element `<INPUT TYPE="CHECKBOX"...>`.

If you want to create a list of check boxes, use WOCheckBoxList instead of this element.

Synopsis

WOCheckBox {value=*defaultValue*; [selection=*selectedValue*;] [name=*fieldName*;] [disabled=*aBoolean*;] ... }

WOCheckBox {checked=*aBoolean*; [name=*fieldName*;] [disabled=*aBoolean*;] ... }

Bindings

value

Value of this input element. If not specified, WebObjects provides a default value.

WOCheckBox

selection

If *selection* and *value* are equal when the page is generated, the check box is checked. When the page is submitted, selection is assigned the value of the check box.

checked

During page generation, if *checked* evaluates to `true`, the check box appears in the checked state. During request handling, *checked* reflects the state the user left the check box in: `true` if checked; `false` if not.

name

Name that uniquely identifies this element within the form. You may specify a name or let WebObjects automatically assign one at runtime.

disabled

If *disabled* evaluates to `true`, this element appears in the page but is not active. That is, *selection* won't contain the user's selection when the page is submitted.

WOCheckBoxList

WOCheckBoxList displays a list of check boxes. The user may select several of the objects in the list, and this sublist is returned as selections.

You should provide the title of a checkbox in `displayString` rather than in `value`. If there is no binding for `displayString`, the string assigned to value is used to identify the checkbox.

Synopsis

WOCheckBoxList { list=*anObjectList*; item=*anIteratedObject*; displayString=*displayedValue*; [value=*aValue*;] [index=*aNumber*;] [prefix=*prefixString*;] [suffix=*suffixString*;] [selections=*selectedValues*;] [name=*fieldName*;] [disabled=*aBoolean*;] [escapeHTML=*aBoolean*;]... }

Bindings

list

Array of objects that the WOCheckBoxList will iterate through.

item

Current item in the list array. (This attribute's value is updated with each iteration.)

displayString

String to display beside the check box for the current item.

value

Value for the `INPUT` tag of the current item (`INPUT type="Checkbox" value=someValue`). You can use this binding as an additional identifier of the item.

index

Index of the current iteration of the WOCheckBoxList.

prefix

An arbitrary HTML string inserted before each value.

suffix

An arbitrary HTML string inserted after each value.

selections

An array of objects that the user chose from the list.

name

Name that uniquely identifies this element within the form. You may specify a name or let WebObjects automatically assign one at runtime.

disabled

If `disabled` evaluates to `true` (or `YES`), this element appears in the page but is not active.

escapeHTML

If `escapeHTML` evaluates to `true` (or `YES`), the string rendered by `displayString` is converted so that characters which would be interpreted as HTML control characters become their escaped equivalent (this is the default). Thus, if a your `displayString` is "`a <b>bold</b> idea`", the string passed to the client browser would be "`a <B>bold</B> idea`", but it would display in the browser as "`a <b>bold</b> idea`". If `escapeHTML` evaluates to `false` (or `NO`), WebObjects simply passes your data to the client browser "as is." In this case, the above example would display in the client browser as "a **bold** idea". If you are certain that your strings have no characters in them which might be interpreted as HTML control characters, you get better performance if you set `escapeHTML` to `false` (or `NO`).

WOComponentContent

WOComponentContent allows you to write nested components as HTML container elements: Elements that can include text and other elements between their opening and closing tags. Using WOComponentContent you can, for example, write a component that defines the header and footer for all of your application's pages.

The WOComponentContent dynamic element doesn't have any attributes. It's simply a marker that specifies where the contents wrapped by the component's `<WEBOBJECT>` tag should go.

Note: You can only have one WOComponentContent element in a given component.

Synopsis

WOComponentContent { }

Example

To write a component that defines the header and footer for some or all of your application's pages, first define a component with HTML similar to the following:

```
<HTML>
```

```
    <HEAD>
        <TITLE>Cool WebObjects App</TITLE>
    </HEAD>
    <BODY>
    <!-- A banner common to all pages here -->
    <!-- Start of content defined by the parent element -->
    <WEBOBJECT name=ParentContent></WEBOBJECT>
    <!-- End of content defined by the parent element -->
    <!-- Put a footer common to all pages here. -->
    </BODY>
</HTML>
```

The <WEBOBJECT> element above is a WOComponentContent element declared like this:

```
ParentContent : WOComponentContent {};
```

To use this component, wrap the contents of all of your other components with a `<WEBOBJECT>` tag that specifies the component defined above. For example, suppose you named the above component `HeaderFooterPage.wo`. You could use it in another component like this:

```
<!-- HTML for a simple component wrapped with HeaderFooterPage -->
<WEBOBJECT name = templateWrapperElement>
    <P>Hello, world!</P>
</WEBOBJECT>
```

Where `templateWrapperElement` is declared in the `.wod` file like this:

```
templateWrapperElement : HeaderFooterPage {};
```

At runtime, the contents wrapped by `templateWrapperElement` are substituted for the WOComponentContent definition. As a result, the HTML generated for this component would be:

```
<HTML>
    <HEAD>
        <TITLE>Cool WebObjects App</TITLE>
    </HEAD>
    <BODY>
    <!-- A banner common to all pages here -->
    <!-- Start of content defined by the parent element -->
```

```
    <P>Hello, world!</P>
    <!-- End of content defined by the parent element -->
    <!-- Put a footer common to all pages here. -->
    </BODY>
</HTML>
```

WOComponentContent

CHAPTER 10

WOConditional

A WOConditional object controls whether a portion of the HTML page will be generated, based on the evaluation of its assigned condition.

Synopsis

WOConditional { condition=*aBoolean*; [negate=*aBoolean*;]}

Bindings

condition

If `condition` evaluates to `true`, and assuming that `negate` is `false`, the contents of the WOConditional are displayed (the portion of the component within the WOConditional is generated).

negate

Inverts the sense of the condition. By default, `negate` is assumed to be `false`.

Example

The negate attribute lets you use the same test to display mutually exclusive information; for example:

HTML File

```
<HTML>
<WEBOBJECTS NAME="PAYING_CUSTOMER">Thank you for your order!</WEBOBJECTS>
<WEBOBJECTS NAME="WINDOW_SHOPPER">Thanks for visiting!</WEBOBJECTS>
</HTML>
```

Declarations File

```
PAYING_CUSTOMER: WOConditional {condition=payingCustomer;};
WINDOW_SHOPPER: WOConditional {condition=payingCustomer; negate=YES;};
```

WOEmbeddedObject

A WOEmbeddedObject provides support for Netscape plug-ins. It corresponds to the HTML element `<EMBED SRC = >`. If the embedded object's content comes from outside the WebObjects application, use the `src` attribute. If the embedded object's content is returned by a method within the WebObjects application, use the `filename` attribute or the `data` and `mimeType` attributes.

Synopsis

WOEmbeddedObject {value=*aMethod*; | src=*aURL*; | filename= *imageFileName*; [framework = *frameworkBaseName*|"app";] | data=*dataObject*; mimeType=*typeString*; [key=*cacheKey*;]... }

Bindings

value

The content for this embedded object in the form of a `WOElement` object. This data can come from a database, a file, or memory.

src

URL containing the embedded object. Use this attribute for complete URLs; for relative URLs use `filename` instead.

filename

Path to the embedded object relative to the WebServerResources directory.

framework

Framework that contains the embedded object. This attribute is only necessary if the object is in a different location from the component. That is, if the component and the embedded object are both in the application or if the component and the embedded object are both in the same framework, this attribute isn't necessary. If the embedded object is in a framework and the component is in an application, specify the framework's name here minus the `.framework` extension. If the embedded object should be in the application but the component is in a framework, specify the "app" keyword in place of the framework name.

data

Specifies any resource in the form of an `NSData`; this data can come from a database, a file, or memory. If you specify resource data, you must specify a MIME type.

mimeType

A string designating a MIME resource type, such as "`image/gif`"; this type tells the client what to do with data. If you provide `data` but no MIME type, WebObjects will raise.

key

A string that functions as a key for caching the data specified in data. If you do not provide a key, the data object is fetched each time it is needed. For further information, see the reference documentation for WOResourceManager, particularly that for the `flushDataCache` method.

CHAPTER 12

WOFileUpload

A WOFileUpload element displays a form element in which a client browser can specify a file to be uploaded to the server. It corresponds to the HTML: `<INPUT type=file>`.

WOFileUpload elements inside of a WOForm require that the WOForm have the attribute's encoding type set as follows:

```
enctype = "multipart/form-data"
```

For further information on the file upload specification, see RFC1867: http://www.w3.org/RT/REC-html32.html#rfc1867.

If you want to process a file upload in a direct action, use `WORequest`'s `formValueForKey` method to get the contents of the file that has been uploaded. This method is declared as follows:

```
public java.lang.Object formValueForKey(java.lang.String aKey)
```

Synopsis

WOFileUpload { filePath=*aPath*; data=*fileData* }

Bindings

filePath

The full file path and name of the file uploaded is sent by the browser and returned as a string to the variable or method bound to this attribute.

data

The file that is uploaded will be returned as an `NSData` object to the variable or method bound to this attribute.

CHAPTER 13

WOForm

A WOForm is a container element that generates a fill-in form. It gathers the input from the input elements it contains and sends it to the server for processing. WOForm corresponds to the HTML element `<FORM ... > ... </FORM>`.

Synopsis

WOForm { [action=*aMethod*; | href=*aURL*;] [multipleSubmit=*aBoolean*;] ... }

Bindings

href

URL specifying where the form will be submitted.

action

Action method that's invoked when the form is submitted. If the form contains a dynamic element that has its own action (such as a WOSubmitButton or a WOActiveImage), that action is invoked instead of the WOForm's.

multipleSubmit

If `multipleSubmit` evaluates to `true`, the form can have more than one WOSubmitButton, each with its own action. By default, WOForm supports only a single WOSubmitButton.

Note: Some older browsers support only a single submit button in a form.

CHAPTER 14

WOFrame

WOFrame represents itself as a dynamically generated Netscape frame element.

Synopsis

WOFrame { value=*aMethod*; | src=*aURL*; | pageName=*aString*; | directActionName=*anActionName*; actionClass=*className*;... }

Bindings

value
: Method that will supply the content for this frame.

src
: External source that will supply the content for this frame.

pageName
: Name of WebObjects page that will supply the content for this frame.

directActionName
: The name of the direct action method (minus the "Action" suffix) that will supply the content for the frame.

actionClass
: The name of the class in which the method designated in `directActionName` can be found. Defaults to "`DirectAction`".

CHAPTER 14

WOFrame

WOGenericContainer

WOGenericContainer supports development of reusable components that closely model the behavior of common HTML elements. For example, along with WOComponentContent, you can use WOGenericContainer to implement your own hyperlink element as a reusable component. WOGenericContainer has attributes that support the `takeValuesFromRequest` and `invokeAction` phases of the component-action request-response loop.

Synopsis

WOGenericContainer { elementName = *aConstantString*; [omitTags=*aBoolean*;] [elementID=*identifier*;] [otherTagString=*aString*;] [formValue=*singleValue*;] [formValues=*arrayOfValues*;] [invokeAction=*aMethod*;]... }

Bindings

elementName

Name of the HTML tag. This name (for example "`TEXTAREA`") will be used to generate the container's opening and closing tags (`<TEXTAREA>...</TEXTAREA>`). `elementName` can either be a constant or a variable, such as a key path. You can also set the value of this attribute

to or `null`, which effectively shuts off this element (that is, WebObjects doesn't generate HTML tags for this element). Alternatively, you can use the `omitTags` attribute to achieve the same effect.

omitTags

Specifies whether the element's tags should be displayed. This attribute is useful for defining an element that conditionally wraps HTML in a container tag. The default value is `false`. If `omitTags` is `true`, the contents of the tag are rendered but not the tags themselves. Using `omitTags` for a container makes the container itself optional.

elementID

Allows programmatic access to the element's element ID. This is a read-only attribute.

otherTagString

Enables any string to be part of the opening tag. This permits standalone attributes such as "checked" or "selected" to be part of a tag.

formValue, formValues

Enables implementation of input-type elements (for example, WOTextField). Bind these attributes to a variable that can contain the component's input value. During the `takeValuesFromRequest` phase, if the element ID of the current generic container matches an element ID of a form value in the request, the form value is pushed into the component using this attribute. The `formValue` attribute corresponds to `WORequest`'s `formValueForKey` while the `formValues` attribute corresponds to `WORequest`'s `formValuesForKey` method; in other words, `formValue` pushes a single attribute while `formValues` pushes an array of attributes.

invokeAction

Enables implementation of action elements (for example, WOHyperlink). During the `invokeAction` phase, if the element ID of the current generic container matches the sender ID of the URL, the method bound to this attribute is evaluated. Just as with any action method, it must return an object that conforms to the `WOActionResults` interface, such as `WOComponent` or `WOResponse`.

CHAPTER 16

WOGenericElement

WOGenericElement supports development of reusable components that closely model the behavior of common HTML elements. For example, you can use WOGenericElement to implement your own image (IMG) element as a reusable component. WOGenericElement has attributes that support the `takeValuesFromRequest` and `invokeAction` phases of the component-action request-response loop.

Synopsis

WOGenericElement { elementName = *aConstantString*; [omitTags=*aBoolean*;] [elementID=*identifier*;] [otherTagString=*aString*;] [formValue=*singleValue*;] [formValues=*arrayOfValues*;] [invokeAction=*aMethod*;]... }

Bindings

elementName

Name of the HTML tag. This name (for example "`HR`") will be used to generate the element's tag (<HR>). `elementName` can either be a constant or a variable, such as a key path. You can also set the value of this attribute to `null`, which effectively shuts off this element (that is, WebObjects doesn't generate HTML tags for this element). Alternatively, you can use the `omitTags` attribute to achieve the same effect.

omitTags

Specifies whether the element's tag should be displayed. The default value is `false` . If `omitTags` is `true` , the entire element is not rendered.

elementID

Allows access to the element's element ID. This is a read-only attribute.

otherTagString

Enables any string to be part of the opening tag. This permits standalone attributes such as "`checked`" or "`selected`" to be part of a tag.

formValue, formValues

Enables implementation of input-type elements (for example, WOTextField). Bind these attributes to a variable that can contain the component's input value. During the `takeValuesFromRequest` phase, if the element ID of the current generic element matches an element ID of a form value in the request, the form value is pushed into the component using this attribute. The `formValue` attribute corresponds to `WORequest`'s `formValueForKey` while the `formValues` attribute corresponds to `WORequest`'s `formValuesForKey` method; in other words, `formValue` pushes a single attribute while `formValues` pushes an array of attributes.

invokeAction

Enables implementation of action elements (for example, WOHyperlink). During the `invokeAction` phase, if the element ID of the current generic element matches the sender ID of the URL, the method bound to this attribute is evaluated. Just as with any action method, it must return an object that conforms to the `WOActionResults` interface, such as `WOComponent` or `WOResponse`.

WOHiddenField

A WOHiddenField adds hidden text to the HTML page. It corresponds to the HTML element `<INPUT TYPE="HIDDEN"...>`. Hidden fields are sometimes used to store application state data in the HTML page.

Synopsis

WOHiddenField { value=*defaultValue*; [name=*fieldName*;] [disabled=*aBoolean*;] ... }

Bindings

value

Value for the hidden text field.

name

Name that uniquely identifies this element within the form. You may specify a name or let WebObjects automatically assign one at runtime.

disabled

If disabled evaluates to `true`, the element appears in the page but is not active.

WOHiddenField

CHAPTER 18

WOHyperlink

WOHyperlink generates a hypertext link in an HTML document.

Synopsis

WOHyperlink { action=*aMethod* | href=*aURL*; | pageName=*aString*; | directActionName=*anActionName*; actionClass=*className*; [fragmentIdentifier=*anchorFragment*;] [string=*aString*;] [target=*frameName*;] [disabled=*aBoolean*;] [secure=*aBoolean;*]...}

Bindings

action
: Action method to invoke when this element is activated. The method must return a WOElement.

href
: URL to direct the browser to when the link is clicked.

pageName
: Name of WebObjects page to display when the link is clicked.

directActionName
: The name of the direct action method (minus the "Action" suffix) to invoke when this element is activated. Defaults to “default”.

WOHyperlink

actionClass

The name of the class in which the method designated in `directActionName` can be found. Defaults to DirectAction.

fragmentIdentifier

Named location to display in the destination page.

string

Text displayed to the user as the link. If you include any text between the `<WEBOBJECT ...>` and `</WEBOBJECT>` tags for this element, the contents of string is appended to that text.

target

Frame in a frameset that will receive the page returned as a result of the user's click.

disabled

If evaluates to `true` , the content string is displayed, but the hyperlink is not active.

secure

Changes the URL prefix from `http` to `https` when WebObjects generates URLs for component actions and direct actions for this element. For this attribute to have any effect, you must provide bindings either for the *action*, *directAction*, *actionClass*, or *pageName* attribute (respecting the valid combinations).

WOImage

A WOImage displays an image in the HTML. It corresponds to the HTML element `<IMG SRC="URL">`.

Synopsis

WOImage { src=*aURL*; | value=*imageData*; | filename= *imageFileName*; [framework = *frameworkBaseName* | "app" ;] | data=*dataObject*; mimeType=*typeString*; [key=*cacheKey*;]... }

Bindings

src

URL containing the image data. Use this attribute for complete URLs; for relative URLs use `filename` instead.

value

Image data in the form of a WOElement object. This data can come from a database, a file, or memory.

filename

Path to the image relative to the WebServerResources directory.

WOImage

framework

Framework that contains the image file. This attribute is only necessary if the image file is in a different location from the component. That is, if the component and the image file are both in the application or if the component and the image file are both in the same framework, this attribute isn't necessary. If the image file is in a framework and the component is in an application, specify the framework's name here (minus the `.framework` extension). If the image file should be in the application but the component is in a framework, specify the `"app"` keyword in place of the framework name.

data

Specifies an image resource in the form of an NSData; this data can come from a database, a file, or memory. If you specify resource data, you must specify a MIME type.

mimeType

A string designating a MIME resource type, such as "`image/gif`", to be put in the content-type header; this type tells the client what to do with data. If you provide `data` but no MIME type, WebObjects throws an exception.

key

A string that the application uses as a key for caching the data specified in `data`. If you do not provide a key, the data object must be fetched each time it is needed. For further information, see the reference documentation for the `WOResourceManager` class, particularly that for the `flushDataCache` method.

WOImageButton

WOImageButton is a graphical submit button. Clicking the image generates a request and submits the enclosing form's values. You often use WOImageButton when you need more than one submit button within a form.

Synopsis

WOImageButton { filename=*anImageName*; [framework=*aFrameworkName* | "app";] | src=*aURL*; | value=*aMethod*; action=*aMethod*; | data=*dataObject*; mimeType=*typeString*; [key=*cacheKey*;] [imageMapFile=*aString*;] [name=*aString*;] [x=*aNumber*; y=*aNumber*;] [disabled=*aBoolean*;] ... }

Bindings

filename

Path to the image relative to the WebServerResources directory.

framework

Framework that contains the image file. This attribute is only necessary if the image file is in a different location from the component. That is, if the component and the image file are both in the application or if the component and the image file are both in the same framework, this attribute isn't necessary. If the image file is in a framework and the component is in an application, specify the framework's name here

(minus the `.framework` extension). If the image file should be in the application but the component is in a framework, specify the `"app"` keyword in place of the framework name.

src

URL containing the image data. Use this attribute for complete URLs; for relative URLs use `filename` instead.

value

Image data in the form of a `WOElement` object. This data can come from a database, a file, or memory.

action

Action method to invoke when this element is clicked.

data

Specifies an image resource in the form of an `NSData`; this data can come from a database, a file, or memory. If you specify resource data, you must specify a MIME type.

mimeType

A string designating a MIME resource type, such as "`image/gif`", to be put in the content-type header; this type tells the client what to do with data. If you provide `data` but no MIME type, WebObjects throws an exception.

key

A string that the application uses as a key for caching the data specified in `data`. If you do not provide a key, the data object must be reloaded each time it is needed. For further information, see the reference documentation for the `WOResourceManager` class, particularly that for the `flushDataCache` method.

imageMapFile

Name of the image map file. See the WOActiveImage description for more information.

name

Name that uniquely identifies this element within the form. You may specify a name or let WebObjects automatically assign one at runtime.

x, y

If specified, returns the coordinates of the user's click within the image.

disabled

If `disabled` evaluates to `true` , the element generates a static image (`<IMG>`) instead of an active image.

CHAPTER 21

WOJavaScript

WOJavaScript lets you embed a script written in JavaScript in a dynamically generated page.

Synopsis

WOJavaScript { scriptFile=*aPath*; | scriptString=*aString*; | scriptSource=*aURL*; [hideInComment=*aBoolean*;] ... }

Bindings

scriptFile

Path to the file containing the script. The path can be statically specified in the declaration file or it can be a `java.lang.String`, an object that responds to a description message by returning a `java.lang.String`, or a method that returns a `java.lang.String`.

scriptString

String containing the script. Typically, `scriptString` is n `java.lang.String` object, an object that responds to a description message by returning a`java.lang.String`, or a method that returns an `java.lang.String`.

scriptSource

URL specifying the location of the script.

hideInComment

If `hideInComment` evaluates to `true` (or `YES`), the script will be enclosed in an HTML comment (`<!-- script //-->`). Since scripts can generate errors in some older browsers that weren't designed to execute them, you may want to enclose your script in an HTML comment. Browsers designed to run these scripts will still be able to execute them despite the surrounding comment tags.

WONestedList

WONestedList recursively displays a hierarchical, ordered (numbered) or unordered (bulleted) list of hyperlinks. This element is useful when you want to display hierarchical lists. When the user clicks one of the objects in the list, it is returned in `selection` and the action method is invoked.

At any point during iteration of the list, the method specified by the `sublist` attribute returns the current list's sublist (if any), `level` specifies the current nesting level (where the topmost level is zero), `index` gives index of the current item within that nesting level (`item` returns the actual item), and `isOrdered` specifies whether the current sublist should be a numbered list or a bulleted list.

Synopsis

WONestedList { list=*anObjectList*; item=*anIteratedObject*; displayString=*displayedValue*; sublist = *aSubarray*; action=*aMethod*; selection=*selectedValue*; [index=*aCurrentIndex*;] [level=*aCurrentLevel*;] [isOrdered=*aBoolean*;] [prefix=*prefixString*;] [suffix=*suffixString*;] [escapeHTML=*aBoolean*;]}

Bindings

list

Hierarchical array of objects that the WONestedList iterates through.

item

Current item in the list array. (This attribute's value is updated with each iteration.)

displayString

String to display as a hyperlink for the current item.

sublist

Method that returns the sublist of the current item or `null` if the current item is a leaf.

action

Action method to invoke when the element is activated. This method must return a `WOElement`.

selection

When the page is submitted, selection contains the item that the user clicked.

index

Index of the current iteration of the WONestedList. The index is unique to each level—that is, it starts at 0 for each sublist.

level

Nesting level of the current iteration of the WONestedList. The topmost level is level 0.

isOrdered

If `isOrdered` evaluates to `true`, the current sublist is rendered as an ordered list. The default is to render as an unordered list.

prefix

An arbitrary HTML string inserted before each value.

suffix

An arbitrary HTML string inserted after each value.

WONestedList

escapeHTML

If `escapeHTML` evaluates to `true`, the string rendered by `displayString` is converted so that characters which would be interpreted as HTML control characters become their escaped equivalent (this is the default). Thus, if a your `displayString` is "`a <b>bold</b> idea`", the string passed to the client browser would be "`a <B>bold</B> idea`", but it would display in the browser as "`a <b>bold</b> idea`". If `escapeHTML` evaluates to `false` , WebObjects simply passes your data to the client browser "as is." In this case, the above example would display in the client browser as "`a bold idea`". If you are certain that your strings have no characters in them which might be interpreted as HTML control characters, you get better performance if you set `escapeHTML` to `false` .

WONestedList

CHAPTER 23

WOParam

WOParam elements are used for passing WOApplet parameters.

Synopsis

WOParam { name=*aString*; value=*aString*; | action=*aMethod*; ... }

Bindings

name
: Symbolic name associated with this element's value.

value
: Value of this parameter.

action
: Method that sets the parameter's value. Use this attribute instead of `value` if you want the parameter to be a WebObjects component.

CHAPTER 23

WOParam

WOPasswordField

A WOPasswordField represents itself as a text field that doesn't echo the characters that a user enters. It corresponds to the HTML element `<INPUT TYPE="PASSWORD"...>`.

Synopsis

WOPasswordField { value=*defaultValue*; [name=*fieldName*;] [disabled=*aBoolean*;] ... }

Bindings

value

During page generation, `value` sets the default value of the text field. This value is not displayed to the user. During request handling, `value` holds the value the user entered into the field, or the default value if the user left the field untouched.

name

This name uniquely identifies this element within the form. You may specify a name or let WebObjects automatically assign one at runtime.

disabled

If `disabled` evaluates to `true`, the element appears in the page but is not active. That is, `value` does not contain the user's input when the page is submitted.

WOPasswordField

WOPopUpButton

WOPopUpButton, when clicked, displays itself as a selection list that allows the user to select only one item at a time. The related element WOBrowser is similar to WOPopUpButton except that it allows the user to select more than one item at a time.

You should provide the title of an item in `displayString` rather than in `value`. If there is no binding for `displayString`, the string assigned to `value` is used for the item.

Synopsis

WOPopUpButton { list=*anArray*; item=*anItem*; displayString=*displayedValue*; [value=*optionValue*;] [selection=*theSelection*; | selectedValue=*selectedValue*;] [name=*fieldName*;] [disabled=*aBoolean*;] [escapeHTML=*aBoolean*;] [noSelectionString=*aString*;]... }

Bindings

list

Array of objects from which the WOPopUpButton derives its values.

item

Identifier for the elements of the list. For example, `aCollege` could represent an object in a `colleges` array.

displayString

Value to display in the selection list; for example, `aCollege.name` for each `college` object in the list.

value

For each `OPTION` tag within the selection, this is the "value" attribute (that is, `<OPTION value="someValue">`). You can use this binding to specify additional identifiers of each item in the menu.

selection

Object that the user chose from the selection list. For the college example, `selection` would be a `college` object.

selectedValue

Value that is used with direct actions to specify which option in the list is selected.

name

Name that uniquely identifies this element within the form. You can specify a name or let WebObjects automatically assign one at runtime.

disabled

If `disabled` evaluates to `true`, this element appears in the page but is not active. That is, `selection` does not contain the user's selection when the page is submitted.

escapeHTML

If `escapeHTML` evaluates to `true`, the string rendered by `displayString` is converted so that characters which would be interpreted as HTML control characters become their escaped equivalent (this is the default). Thus, if a your `displayString` is "`a <b>bold</b> idea`", the string passed to the client browser would be "`a &lt;B&gt;bold&lt;/B&gt; idea`", but it would display in the browser as "`a <b>bold</b> idea`". If `escapeHTML` evaluates to `false` , WebObjects simply passes your data to the client browser "as is." In this case, the above example would display in the client browser as "`a bold idea`". If you are certain that your strings have no characters in them which might be interpreted as HTML control characters, you get better performance if you set `escapeHTML` to `false`.

noSelectionString

Enables the first item to be "empty." Bind this attribute to a string (such as an empty string) that, if chosen, represents an empty selection. When this item is selected, the `selection` attribute is set to `null`.

CHAPTER 26

WOQuickTime

WOQuickTime is a dynamic element that you can use to incorporate QuickTime objects (movie, sound, VR, ...) into your WebObjects applications. The WOQuickTime API is essentially based on the QuickTime plug-ins API.

WOQuickTime supports QuickTime VR with hotspots. If you specify a list of hotspots and the user clicks inside the QuickTime VR object, the method specified by the `action` attribute is performed and the `selection` attribute is set to the value of the selected hotspot.

You should use WOQuickTime components outside of an HTML form.

Synopsis

WOQuickTime { filename=*imageFilePath*; | src=*aURL*; |
[framework=*frameworkName* | "app";] width=*anInt*; height=*anInt*;
[hidden=*aBoolean*;] [pluginsPage=*aURL*;] [hotspotList=*arrayOfIDs*;
selection=*aString*; action=*aMethod*; href=*anHREF*; | pageName=*page*;
[target=*frameTarget*;]] [bgcolor=*hexString*;] [volume=*anInt*;] [pan=*panAngle*;]
[tilt=*tiltAngle*;] [fov=*fieldOfView*;] [node=*initialNode*;]
[correction=NONE | PARTIAL | FULL;] [cache=*aBoolean*;] [autoplay=*aBoolean*;]
[playeveryframe=*aBoolean*;] [controller=*aBoolean*;] [prefixhost=*aBoolean*;]}

Bindings

WOQuickTime has the following attributes. Those attributes relevant only to VR movies are indicated with "[VR]" in the description.

filename

Path to the QuickTime object relative to the WebServerResources directory.

src

URL locating the QuickTime object. Use this attribute for complete URLs; for relative URLs use `filename` instead.

framework

The framework that contains the QuickTime object. This attribute is only necessary if the QuickTime object is in a different location from the component. That is, if the component and the QuickTime object are both in the application or if the component and the QuickTime object are both in the same framework, this attribute isn't necessary. If the QuickTime object is in a framework and the component is in the application, specify the framework's name here (minus the `.framework` extension). If the QuickTime object should be in the application but the component is in a framework, specify the "`app`" keyword in place of the framework name.

width

QuickTime object width in pixels. The `width` attribute is required. Never specify a width of less than 2 as this can cause problems with some browsers. If you are trying to hide the movie, use the `hidden` attribute instead. If you don't know the width of the movie, open your movie with QuickTime Player (it comes with QuickTime) and select Show Movie Info from the Window menu. If you don't use the `scale` attribute and you supply a width that is smaller than the actual width of the movie, the movie will be cropped to fit. If you supply a width that is greater than the width of the movie, the movie will be centered inside this width.

height

Quicktime object height in pixels. If you want to display the movie's controller, you'll need to add 16 pixels to the height. `height` is required unless you use the `hidden` attribute. Never specify a height of less than 2 as this can cause problems with some browsers. If you are trying to hide the movie, use the `hidden` attribute instead. If you don't know the height of the movie, open your movie with QuickTime Player and select Show Movie Info from the Window menu. If you do not use the `scale` attribute and you supply a height that is smaller than the actual height of the movie (plus 16 if you are showing the controller), the movie will be cropped to fit. If you supply a height that is greater than the height of the movie, the movie will be centered inside this height.

pluginsPage

This optional attribute allows you to specify a URL from which the user can fetch the necessary plug-in if it is not installed. This attribute is handled by your browser. If your browser cannot find the plug-in when loading your page, it will warn the user and allow them to bring up the specified URL. Generally this parameter should be set to "`http://www.apple.com/quicktime`". This attribute is appropriate for both QuickTime movies and QuickTime VR Objects and Panoramas.

hotspotList

[VR] The hotspot list is an array of strings, each of which should be mapped to a hotspot ID as defined when the hotspots are created with the QuickTime VR authoring tools.

selection

[VR] A string corresponding to the ID of the user-selected hotspot or `null` if none is selected.

action

Method to invoke when the QuickTime object is clicked. The selection parameter then contains the ID of the selected hotspot if a hotspot list has been specified, or `null` otherwise.

href

An optional attribute for specifying a URL to direct the browser to when the QuickTime object is clicked and no hotspots are hit.

pageName

An optional attribute specifying the name of the WebObjects page to display when the QuickTime object is clicked and no hotspots are hit.

bgcolor

Background color for the QuickTime object. This is an optional attribute. Use `bgcolor` to specify the background color for any space that is not taken by the movie—as, for example, if you embed a 160x120 movie in a 200x120 space. Specify the color as a hex value.

target

When set, the `target` attribute is the name of a valid frame (including `_self`, `_top`, `_parent`, `_blank` or an explicit frame name) that will be the target of a link specified by the hotspot or `href` attribute.

volume

An optional attribute affecting the initial volume level. Possible values are 0 through 100. A setting of 0 effectively mutes the audio; a setting of 100 is maximum volume.

pan

[VR] This optional attribute allows you to specify the initial pan angle for a QuickTime VR movie.The range of values for a typical movie would be 0.0 to 360.0 degrees. If no value for `pan` is specified, the value stored in the movie is used.

tilt

[VR] This optional attribute allows you to specify the initial tilt angle for a QuickTime VR movie. The range of values for a typical movie would be -42.5 to 42.5 degrees. If no value for `tilt` is specified, the value stored in the movie is used.

fov

[VR] This optional attribute allows you to specify the initial field of view angle for a QuickTime VR movie.The range of values for a typical movie would be 5.0 to 85.0 degrees. If no value is specified for `fov`, the value stored in the panoramic movie is used.

node

[VR] This optional attribute allows you to specify the initial node for a multi-node QuickTime VR movie. If no value is specified for `node` , the default node and view (specified at creation time of the movie) is used.

correction

[VR] (optional) Possible values are “`NONE`”, “`PARTIAL`”, or “`FULL`” (the default). This attribute is only appropriate for QuickTime VR objects and panoramas.

cache

(optional) If `cache` evaluates to `true`, the browser caches movies when possible just like other documents.

autoplay

(optional) When `autoplay` evaluates to `true`, causes the movie to start playing as soon as the QuickTime plug-in estimates that it'll be able to play the entire movie without waiting for additional data. This attribute's default is specified by a user setting in the QuickTime Plug-in Preferences.

hidden

This optional attribute controls the visibility of the movie. By default the value is `true`; if you set it to `false` the movie won't be visible on the page. This option is not appropriate for QuickTime VR Objects or Panoramas. You can use `hidden` to hide a sound-only movie.

playEveryFrame

When this optional attribute evaluates to `true` the QuickTime plug-in plays every frame, even if it is necessary to play at a slower rate to do so. This attribute is particularly useful to play simple animations, and is appropriate for QuickTime movies. Note that enabling `playEveryFrame` will turn off any audio tracks your movie may have.

controller

This optional attribute sets the visibility of the movie controller. If you don't specify `controller`, the default is `true` for QuickTime movies. For compatibility with existing web pages, the default is `false` for QuickTime VR movies.

prefixHost

This attribute should be used to fix a bug with the QuickTime 2.x plug-in on Windows platforms. Setting `prefixHost` to `true` automatically adds the http host name at the beginning of each dynamic URL, allowing old plug-ins to correctly handle WOQuickTime component. The default is `false` .

CHAPTER 26

WOQuickTime

CHAPTER 27

WORadioButton

WORadioButton represents itself as an on-off switch. Radio buttons are normally grouped, since the most important aspect of their behavior is that they allow the user to select no more than one of several choices. If the user selects one button, the previously selected button (if any) becomes deselected.

Since radio buttons normally appear as a group, WORadioButton is commonly found within a WORepetition. Alternatively, you can use the WORadioButtonList element.

Synopsis

WORadioButton {value=*defaultValue*; [selection=*selectedValue*;] [name=*fieldName*;] [disabled=*aBoolean*;] ... }

WORadioButton {checked=*aBoolean*; [name=*fieldName*;] [disabled=*aBoolean*;] ... }

Bindings

Note: in a WORadioButton declaration you must supply either `checked` or `value`, but not both: they are mutually exclusive.

value

Value of this input element. If not specified, WebObjects provides a default value.

selection

If `selection` and `value` are equal when the page is generated, the radio button is selected. When the page is submitted, `selection` is assigned the value of the radio button.

checked

During page generation, if `checked` evaluates to `true`, the radio button appears in the selected state. During request handling, `checked` reflects the state the user left the radio button in: `true` if checked; `false` if not.

name

Name that identifies the radio button's group. Only one radio button at a time can be selected within a group.

disabled

If `disabled` evaluates to `true` , this element appears in the page but is not active. That is, `selection` does not contain the user's selection when the page is submitted.

WORadioButtonList

WORadioButtonList displays a list of radio buttons. The user may select one of the objects in the list, and this object is returned as selection.

Synopsis

WORadioButtonList { list=*anObjectList*; item=*anIteratedObject*; displayString=*displayedValue*; [value=*aValue*;] [index=*aNumber*;] [prefix=*prefixString*;] [suffix=*suffixString*;] [selection=*selectedValue*;] [name=*fieldName*;] [disabled=*aBoolean*;] [escapeHTML=*aBoolean*;]... }

Bindings

Note: You should provide the title of a radio button in `displayString` rather than in `value`. If there is no binding for `displayString`, the string assigned to `value` is used as the label of the button.

list

Array of objects that the WORadioButtonList will iterate through.

item

Current item in the list array. (This attribute's value is updated with each iteration.)

displayString

String to display beside the radio button for the current item.

value

Value for the `INPUT` tag of the current item (`INPUT type="RadioButton" value="someValue"`>.

index

Index of the current iteration of the WORadioButtonList.

prefix

An arbitrary HTML string inserted before each value.

suffix

An arbitrary HTML string inserted after each value.

selection

An object that the user chose from the list.

name

Name that uniquely identifies this element within the form. You may specify a name or let WebObjects automatically assign one at runtime.

disabled

If `disabled` evaluates to `true`, this element appears in the page but is not active.

escapeHTML

If `escapeHTML` evaluates to `true`, the string rendered by `displayString` is converted so that characters which would be interpreted as HTML control characters become their escaped equivalent (this is the default). Thus, if a your `displayString` is "`a <b>bold</b> idea`", the string passed to the client browser would be "`a <B>bold</B> idea`", but it would display in the browser as "`a <b>bold</b> idea`". If `escapeHTML` evaluates to `false` , WebObjects simply passes your data to the client browser "as is." In this case, the above example would display in the client browser as "`a bold idea`". If you are certain that your strings have no characters in them which might be interpreted as HTML control characters, you get better performance if you set `escapeHTML` to `false`.

CHAPTER 29

WORepetition

A WORepetition is a container element that repeats its contents (that is, everything between the <WEBOBJECT...> and </WEBOBJECT...> tags in the template file) a given number of times. You can use a WORepetition to create dynamically generated ordered and unordered lists or banks of check boxes or radio buttons.

Synopsis

WORepetition {list=*anObjectList*; item=*anIteratedObject*; [index=*aNumber*;] [identifier=*aString*;] ... };

WORepetition {count=*aNumber*; [index=*aNumber*;]}

Bindings

list

Array of objects that the WORepetition will iterate through. Ideally, this should be an immutable array. If you must pass a mutable array, your code must not alter the array while the WORepetition is iterating through it.

item

Current item in the list array. (This attribute's value is updated with each iteration.)

WORepetition

index

Index of the current iteration of the WORepetition. (This attribute's value is updated with each iteration.

count

Number of times this element will repeat its contents.

CHAPTER 30

WOResetButton

A WOResetButton element generates a reset button in an HTML page. This element is used within HTML forms.

Synopsis

WOResetButton { value=*aString*; ... }

Bindings

value

Title of the button.

WOResetButton

WOResourceURL

WOResourceURL enables the creation of URLs to return resources, such as images and sounds. You can use this element for a variety of purposes, but it is primarily intended to support JavaScript within a WebObjects application.

Synopsis

WOResourceURL { filename= *imageFileName*; [framework = *frameworkBaseName* | "app";] | data=*dataObject*; mimeType=*typeString*; [key=*cacheKey*;]}

Bindings

filename

Path to the resource relative to the WebServerResources directory.

framework

Framework that contains the resource file. This attribute is only necessary if the file is in a different location from the component. That is, if the component and the file are both in the application or if the component and the file are both in the same framework, this attribute isn't necessary. If the resource file is in a framework and the component is in an application, specify the framework's name here (minus the

.framework extension). If the resource file should be in the application but the component is in a framework, specify the "app" keyword in place of the framework name.

data

Specifies any resource in the form of an NSData; this data can come from a database, a file, or memory. If you specify resource data, you must specify a MIME type.

mimeType

A string designating a MIME resource type, such as "image/gif"; this type tells the client what to do with data. If you provide data but no MIME type, WebObjects throws an exception.

key

A string that functions as a key for caching the data specified in data. If you do not provide a key, the data object must be fetched each time it is needed. For further information, see the reference documentation for the WOResourceManager class, particularly that for the flushDataCache method.

CHAPTER 32

WOString

A WOString represents itself in the HTML page as a dynamically generated string.

Synopsis

WOString { value=*aString*; [formatter=*formatterObj*;] [escapeHTML=*aBoolean*;] [dateformat=*dateFormatString*;] [numberformat=*numberFormatString*;] [valueWhenEmpty=*emptyString*;]}

Bindings

value

Text to display in the HTML page. `value` is typically assigned an `java.lang.String` object, an object that responds to a description message by returning an `java.lang.String`, or a method that returns an `java.lang.String`. The `java.lang.String`'s contents are substituted into the HTML in the place occupied by this dynamic element.

escapeHTML

If `escapeHTML` evaluates to `true`, the string rendered by `value` is converted so that characters which would be interpreted as HTML control characters become their escaped equivalent (this is the default). Thus, if your value is "`a <b>bold</b> idea`", the string passed to the client browser would be "`a <B>bold</B> idea`", but it would

display in the browser as "`a <b>bold</b> idea`". If `escapeHTML` evaluates to `false`, WebObjects simply passes your data to the client browser "as is." In this case, the above example would display in the client browser as "`a bold idea`". If you are certain that your strings have no characters in them which might be interpreted as HTML control characters, you get better performance if you set `escapeHTML` to `false`.

formatter

An instance of an `NSFormatter` subclass to be used to format object values for display as strings. This attribute should specify a variable containing (or method returning) a preconfigured formatter object.

If a user enters an "unformattable" value, WOString passes the invalid value through, allowing you to send back an error page that shows the invalid value.

dateformat

A format string that specifies how `value` should be formatted as a date. If a date format is used, `value` can be assigned an NSTimestamp object (if it is assigned an `java.lang.String` object, it is stored as the string representation of an NSTimestamp object). If the element's value can't be interpreted according to the format you specify, `value` is set to `null`. See the NSTimestamp class specification for a description of the date format syntax.

numberformat

A format string that specifies how `value` should be formatted as a number. If the element's value can't be interpreted according to the format you specify, `value` is set to `null`. See the NSNumberFormatter class specification for a description of the number format syntax.

valueWhenEmpty

A string that is substituted for *value* when *value* is the empty string.

CHAPTER 33

WOSubmitButton

A WOSubmitButton element generates a submit button in an HTML page. This element is used within HTML forms.

Synopsis

WOSubmitButton { action=*submitForm*; value=*aString*; [disabled=*aBoolean*;] [name=*aName*;] ... }

Bindings

action

Action method to invoke when the form is submitted.

value

Title of the button.

disabled

If `disabled` evaluates to `true`, the element appears in the page but is not active. That is, clicking the button does not actually submit the form.

name

Name that uniquely identifies this element within the form. You may specify a name or let WebObjects automatically assign one at runtime.

WOSubmitButton

WOSwitchComponent

WOSwitchComponent provides a way to determine at runtime which nested component should be displayed. This component is useful when you want to decide how to display information based on the state of the application.

Synopsis

WOSwitchComponent { WOComponentName=*aComponentName*; ... }

Bindings

WOComponentName

Name of the component to display. This attribute can be a string or a method that returns the name of a component.

If the component specified in `WOComponentName` takes attributes, pass these attributes along to WOSwitchComponent following the `WOComponentName` attribute. Note that this means that all components that can be displayed by this WOSwitchComponent must use the same API.

WOSwitchComponent

WOText

WOText generates a multi-line field for text input and display. It corresponds to the HTML element <TEXTAREA>.

Synopsis

WOText { value=*defaultValue*; [name=*fieldName*;] [disabled=*aBoolean*;] ... }

Bindings

value

During page generation, `value` specifies the text that is displayed in the text field. During request handling, `value` contains the text as the user left it.

name

The name that uniquely identifies this element within the form. You may specify a name or let WebObjects automatically assign one at runtime.

disabled

If `disabled` evaluates to `true`, the text area appears in the page but is not active. That is, `value` does not contain the user's input when the page is submitted.

CHAPTER 35

WOText

WOTextField

A WOTextField represents itself as a text input field. It corresponds to the HTML element `<INPUT TYPE="TEXT"...>`.

Synopsis

WOTextField { value=*aValue*; [formatter=*formatterObj*;] [dateformat=*dateFormatString*;] [numberformat=*numberFormatString*;] [name=*fieldName*;] [disabled=*aBoolean*;] ... }

Bindings

value

During page generation, *value* sets the default value displayed in the single-line text field. During request handling, it holds the value the user entered into the field, or the default value if the user left the field untouched.

formatter

An instance of a `java.text.Format` subclass to be used to format object values for display as strings, and format user-entered strings back into object values. This attribute should specify a variable containing (or method returning) a preconfigured formatter object.

If a user enters an "unformattable" value, WOTextField passes the invalid value through, allowing you to send back an error page that shows the invalid value.

dateformat

A format string that specifies how `value` should be formatted as a date. If a date format is used, `value` can be assigned an NSTimestamp object (if it is assigned a `java.langString` object, it will be stored as the string representation of an NSTimestamp object). If the element's value can't be interpreted according to the format you specify, it is set to `null`. See the NSTimestamp class specification for a description of the date format syntax.

numberformat

A format string that specifies how `value` should be formatted as a number. If the element's value can't be interpreted according to the format you specify, `value` is set to `null`. See the NSNumberFormatter class specification for a description of the number format syntax.

name

Name that uniquely identifies this element within the form. You may specify a name or let WebObjects automatically assign one at runtime.

disabled

If `disabled` evaluates to `true`, the element appears in the page but is not active. That is, `value` does not contain the user's input when the page is submitted.

CHAPTER 37

WOVBScript

WOVBScript lets you embed a script written in Visual Basic in a dynamically generated page.

Synopsis

WOVBScript { scriptFile=*aPath* | scriptString=*aString* | scriptSource=*aURL*; [hideInComment=*aBoolean*;] ... }

Bindings

scriptFile

Path to the file containing the script. The path can be statically specified in the declaration file or it can be a `java.lang.String`, an object that responds to a description message by returning an `java.lang.String`, or a method that returns an `java.lang.String`.

scriptString

String containing the script. Typically, `scriptString` is an `java.lang.String` object, an object that responds to a description message by returning an `java.lang.String`, or a method that returns an `java.lang.String`.

scriptSource

URL specifying the location of the script.

hideInComment

If `hideInComment` evaluates to `true`, the script is enclosed in an HTML comment (`<!-- script -->`). Since scripts can generate errors in some older browsers that weren't designed to execute them, you may want to enclose your script in an HTML comment. Browsers designed to run these scripts will still be able to execute them despite the surrounding comment tags.