Using Dynamic Elements

Dynamic elements are the heart of WebObjects. They transform a static HTML page into one whose contents can be derived from calculation, database search, or some other dynamic means.

Introduction

Simple Dynamic Elements

WOString	Simple character string.
WOImage	Passive image.
WOActiveImage	Active image with hot zones.
WOHyperlink	Hypertext links.
WOConditional	Container element that conditionally displays its contents.
WORepetition	Container element that repeats its contents.

Form-Based Controls

WOForm	Form input element.
WOBrowser	Selection list allowing multiple selection.
WOCheckBox	Check box.
WOHiddenField	Hidden text field.
WOPasswordField	Password text field.
WOPopUpButton	Selection list allowing a single selection.
WORadioButton	Radio button.
WOSubmitButton	Standard submit button.
WOResetButton	Standard reset button.
WOText	Multiline text input area.
WOTextField	Single-line text input area.
WOStateStorage	Element used to store application state data in the HTML page.

Other Dynamic Elements

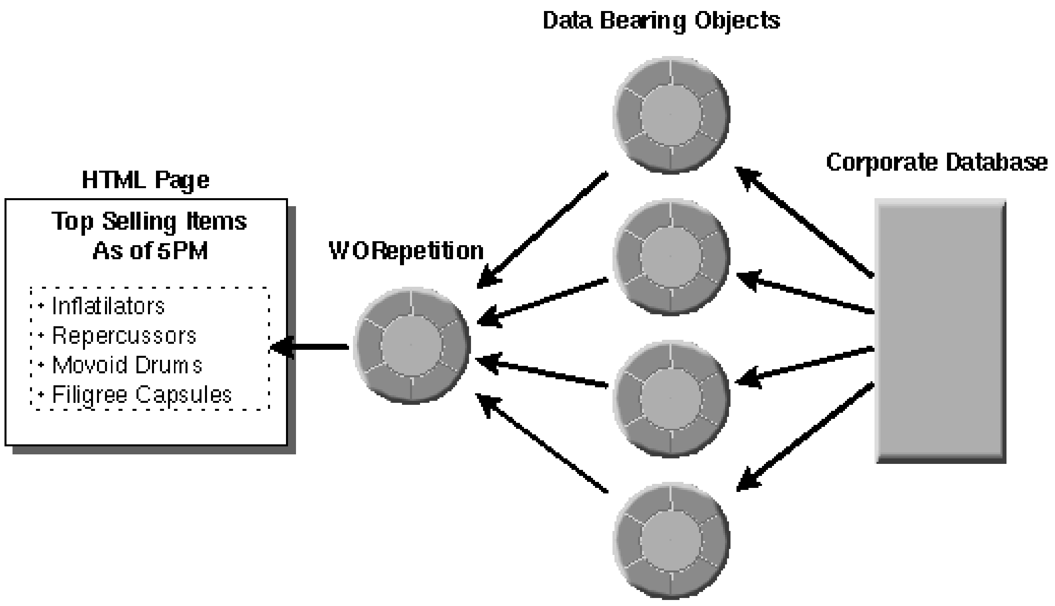
WOGenericElement	Support for unknown HTML elements
WOGenericContainer	Support for unknown HTML container elements
WOFrame	Support for Netscape frames
WOApplet	Support for Java applets.
WOEmbeddedObject	Plug-ins: support for the <EMBED> element from Netscape.

Introduction: Dynamic Elements

A WebObjects dynamic element acts as a bridge between scripted behavior (or behavior defined in compiled code) and an HTML page. A dynamic element can serve as a bridge by being able to:

- Set its attributes to values derived from a method defined in a script or in a custom object.
- Represent itself as HTML when called upon to do so.

Consider the WORepetition dynamic element. A WORepetition can take its values from the results of a database query or some other source and then generate the HTML tags (<UL>, <LI>, etc.) to display the data in a list:



The HTML for this page comes in part from statically declared elements (the heading, for example) and in part from dynamically generated ones (such as each item in the list). The HTML template file for this page shows how the dynamic elements (identified by the tag WEBOBJECT) and the static elements are intermixed:

```
<HTML>
 <H3>Top Selling Items as of 5PM</H3>
 <OL>
     <WEBOBJECT NAME = "ITEMS">
         <LI><WEBOBJECT NAME = "ITEMNAME"></WEBOBJECT></LI>
     </WEBOBJECT>
 </UL>
</HTML>
```

The corresponding declarations file associates each WEBOBJECT tag in the HTML template with a particular dynamic element and indicates how the element’s attributes will be initialized:

```
ITEMS: WORepetition {list=findTopSellers; item=anItem};
ITEMNAME: WOString {value=anItem};
```

Finally, a script file specifies how values used to initialize a dynamic object’s attributes (such as the repetition’s list) are derived. For example,

```
id topSellers;
```

```
- findTopSellers {
  topSellers = /* message to fetch names of top sellers */;
}
```

How Dynamic Elements are Initialized

A dynamic element requires information from the HTML template file, declarations file, and script file to be fully functional, as discussed above. The HTML template file establishes the location of the dynamic element and may provide a template representation for it in the HTML page. If the template file doesn't specify the representation of a dynamic element, the element can create one of its own. For example, in this HTML template the HTML tags in bold are optional:

```
<HTML>
<WEBOBJECT NAME="MYFORM">
<FORM>
  Your Name:
  <WEBOBJECT NAME="TEXTFIELD">
      <INPUT TYPE="text"></WEBOBJECT>
  <INPUT TYPE="submit">
</FORM>
</WEBOBJECT>
</HTML>
```

MYFORM is a WOForm object and TEXTFIELD is a WOTextField object, as specified in the declarations file:

```
MYFORM: WOForm {action = someAction};
INPUTFIELD: WOTextField {value = textFieldValue};
```

Although the HTML tags are optional, it's best to include them in the template file so that, when the template is viewed in a web browser or other HTML rendering tool, it looks as much as possible like the dynamic page that will be generated at runtime.

The declarations file specifies how a dynamic element's attributes will be initialized. For example, this declaration indicates that the dynamic element named TEXTFIELD is a WOTextField object whose input will be returned in the variable "textFieldValue".

```
INPUTFIELD: WOTextField {value=textFieldValue};
```

Each dynamic element declares a set of attributes for you to initialize. However, you can also specify additional attributes and their values within a dynamic element's declaration:

```
TEXTFIELD: WOTextField {value = textFieldValue; size = 4};
```

In this case, only the **value** attribute is required by the WOTextField object. When a dynamic element is asked to produce its HTML representation, these additional attributes and values are simply copied into the HTML stream. The values for these additional attributes can be derived dynamically, just as with the built-in attributes.

WOString

Synopsis

WOString { **value**=*aString*; };

Description

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

value

The text to display in the HTML page. **value** is typically assigned an NSString object, an object that responds to a **description** message by returning an NSString, or a method that returns an NSString.
The NSString’s contents are substituted into the HTML in the place occupied by this dynamic element.

WOImage

Synopsis

WOImage { **src**=*aPath* | **value**=*imageData*;... };

Description

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

src
: The path to the file containing the image data. The source can be statically specified in the declaration file or it can be an NSString, and object that responds to a **description** message by returning an NSString, or a method that returns an NSString.

value
: The the image data in the form of a WOElement object. This data can come from a database, a file, or memory.

WOActiveImage

Synopsis

WOActiveImage { src=*aPath* | **value**=*aMethod*; **action**=*aMethod* | **href**=*aURL*; [**imageMapFile**=*aString*;] [**name**=*aString*;] [**x**=*aNumber*; **y**=*aNumber*;] [**target**=*frameName*;] [**disabled**=YES|NO;] ... **};**

Description

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.

If located outside an HTML form, a WOActiveImage functions as a mapped, active image. When the user clicks such a WOActiveImage, the coordinates of the click are sent back to the server. Depending on where the user click, different actions can be invoked. An image map file associates actions with each of the defined areas of the image.

Within an HTML form, a WOActiveImage functions as a graphical submit button. You typically use WOActiveImages when you need more than one submit button within a form.

src
: The path to the file containing the image data. **src** can be statically specified in the declarations file, an object that responds to a **description** message by returning an NSString, or a method that returns an NSString.

value
: The the image data in the form of a WOElement object. This data can come from a database, a file, or memory.

action
: The 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
: The URL to direct the browser to as a default when the image is clicked and no hotzones are hit.

imageMapFile
: The name of the image map file.

name
: If name is specified then the hit point is specified as **name.x**=*value*;

name.y=*value*; in the form. This is useful for people who want to use imagemap to submit a form to an external URL which expects the hit point to be a certain format.

x, y
: If specified, returns the coordinates of the user’s click within the image.

target
: Specifies the frame in a frameset that will receive the page returned as a result of the user’s click.

disabled
: If YES, a regular image element "<IMG>" is generated,rather than an active image.

The Image Map File

If **imageMapFile** is specified, WebObjects searches for the file within the component bundle (*Component***.wo/**). If it isn’t found there, WebObjects searches the application directory (*MyApplication*/).

Each line in the image map file has this format:

shape action coordinateÿlist

shape
: Either ’rect’ or ’circle’ (polygon not yet supported). For ’rect’ shape, the coordinates x1,y1 specify the upperÿleft corner of the hot zone, and x2,y2 specify lower right corner. For ’circle’ shape, the x1,y1 is the origin, and x2,y2 is a point on the circle.

action
: The name of the method to invoke.

coordinateÿlist
: x1,y1 x2,y2 ...

Here’s an example of an image map file:

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

WOHyperlink

Synopsis

WOHyperlink { **action**=*aMethod* | **href**=*aURL* | **pageName**=*aString*; [**string**=*aString*;] [**target**=*frameName*;] [**disabled**=YES|NO;] ... };

Description

WOHyperlink generates a hypertext link in an HTML document.

action
: The action method to invoke when this element is activated.

href
: The URL to direct the browser to when the image is clicked.

pageName
: Specifies the WebObjects page name to return when the link is clicked.

string
: Specifies the text displayed to the user as the link&emdash;the text between <A> and </A>.

target
: Specifies the frame in a frameset that will receive the page returned as a result of the user's click.

disabled
: If evaluates to YES, the content string is displayed, but the hyperlink is not active.

WOConditional

Synopsis

WOConditional { condition = YES|NO; **};**

Description

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

condition

The expression to evaluate. If the expression evaluates to YES, the HTML code controlled by the WOConditional object is emitted; otherwise it is not.

WORepetition

Synopsis

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

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

Description

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.

list
    The array of objects through which the WORepetition will iterate.
item
    The current item in the **list** array.
identifier
    The value used to uniquely identify this item in the **list** array. Typically it is the primary key of an enterprise object.
count
    The number of times this element will repeat its contents.
index
    The index of the current iteration of the WORepetition.

WOForm

Synopsis

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

Description

A WOForm is a container element that generates a fillÿin 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>.

href
: The URL specifying where the form will be submitted.

action
: The action method that’s invoked when the form is submitted. If the form contains an WebObject that has its own action (such as a WOSubmitButton or a WOActiveImage), that action is invoked instead of the WOForm’s.

WOBrowser

Synopsis

WOBrowser { **list**=*anArray*; [**item** = *anItem*; **value** = *displayedValue*;] [**selections** = *objectArray*;] [**name** = *fieldName*;] [**disabled** = YES|NO;] ... **};**

Description

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

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

item
: The identifier for the elements of the list. For example, **aCollege** could represent an object in the colleges array.

value
: The value to display in the selection list; for example, **aCollege.name** for each college object in the list.

selections
: An array of objects that the user chose from **list**. For the college example, **selections** would hold college objects.

name
: A 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 YES, this element appears in the page but is not active.

WOCheckBox

Synopsis

WOCheckBox { [**name** = *fieldName*;] [**value** = *defaultValue*; [**selection** = *selectedValue*;]] [**disabled** = YES|NO;] ... **};**

WOCheckBox { [**name** = *fieldName*;] [**checked** = YES|NO;] [**disabled** = YES|NO;] ... **};**

Description

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"...>.

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

value
: Sets the 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 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 YES, the check box appears in the checked state. During request handling, **checked** reflects the state the user left the check box in: YES if checked; NO if not.

disabled
: If **disabled** evaluates to YES, this element appears in the page but is not active.

WOHiddenField

Synopsis

WOHiddenField { [**name** = *fieldName*;] **value** = *defaultValue*; [**disabled** =YES|NO;] ... **};**

Description

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. In WebObjects, the WOStateStorage element is designed expressly for this purpose.

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

value
: Sets the value for the hidden text field.

disabled
: If **disabled** evaluates to YES, the element appears in the page but is not active.

WOPasswordField

Synopsis

WOPasswordField { [**name** = *fieldName*;] **value** = *defaultValue*; [**disabled** =YES|NO;] ... **};**

Description

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"...>.

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

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.

disabled
: If **disabled** evaluates to YES, the element appears in the page but is not active.

WOPopUpButton

Synopsis

WOPopUpButton { **list**=*anArray*; [**item** = *anItem*; **value** = *displayedValue* ;] [**selection** = *objectArray*;] [**name** = *fieldName*;] [**disabled** = YES|NO;] ... **};**

Description

WOPopUpButton 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.

list
: An array of objects from which the WOPopUpButton derives its values. For example, **colleges** could name the array containing objects that represent individual schools.

item
: The identifier for the elements of the list. For example, **aCollege** could represent an object in the colleges array.

value
: The value to display in the selection list; for example, **aCollege.name** for each college object in the list.

selection
: An array of objects that the user chose from the selection list. For the college example, **selection** would hold college objects. Since a WOPopUpButton lets the user select only one item at a time, this array holds no more than one item.

name
: A 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 YES, this element appears in the page but is not active.

WORadioButton

Synopsis

WORadioButton {[**name** = *fieldName*;] [**value** = *defaultValue*; [**selection** = *selectedValue*]]; [**disabled** = YES|NO;] ... **};**

WORadioButton {[**name** = *fieldName*;] [**checked** = YES|NO;] [**disabled** = YES|NO;] ... **};**

Description

WORadioButton represents itself as an onÿoff 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.

name

A name that identifies the radio button’s group. Only one radio button at a time can be selected within a group.

checked

During page generation, if **checked** evaluates to YES, the radio button appears in the selected state. During request handling, **checked** reflects the state the user left the radio button in: YES if checked; NO if not.

value

Sets the 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.

disabled

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

Note that either **checked** or **value** is required in a WORadioButton declaration, but that they are mutually exclusive.

WOSubmitButton

Synopsis

WOSubmitButton { action = *submitForm*; **value** = *aString*; [**disabled** = YES|NO;] [**name** = *aName*;] **};**

Description

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

action
    The action method to invoke when the form is submitted.
value
    The title of the button.

Some browsers permit multiple submit buttons in a single form. For multiple submit buttons in a form, each WOSubmitButton must have a unique **value** attribute and **action** attribute. In general, however, it’s better to implement multiple submit buttons using WOActiveImage elements.

WOResetButton

Synopsis

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

Description

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

value

The title of the button.

WOText

Synopsis

WOText{ value = *defaultValue*; [**disabled** = YES|NO;] [**name** = *fieldName* ;] ... **};**

Description

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

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.

disabled

If **disabled** evaluates to YES, the text area appears in the page but no input is allowed.

name

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

WOTextField

Synopsis

WOTextField {[**name** = *fieldName*;] **value** = *defaultValue*; [**disabled** =YES|NO;] ... **};**

Description

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

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

value
: During page generation, **value** sets the default value displayed in the singleÿline 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.

disabled
: If **disabled** evaluates to YES, the element appears in the page but is not active.

WOStateStorage

Synopsis

WOStateStorage { [**size**=*numBytes*;] };

Description

A WOStateStorage element provides a simple mechanism for storing application state in an HTML page. If you include a WOStateStorage element in a form, any session and persistent data will be stored in the page rather than on the server.

WOStateStorage uses HTML hidden fields (<INPUT TYPE="HIDDEN"...>) to store state data. It will use as many hidden field as needed to store the data, but no field will be larger than the size specified by the **size** attribute. The default size setting is designed to work with most browsers.

size
: The maximum size for each of the hidden fields used to store the state data. This attribute is optional; if **size** is not specified, the maximum size for hidden fields will be 1000 bytes.

Since WOStateStorage elements are implemented using hidden fields-which in HTML must be located within a form-they too must be located within a form. If a page has more than one form, you must declare a WOStateStorage element within each form.

WOGenericElement

Synopsis

WOGenericElement { elementName = *aConstantString*; ... **};**

Description

WOGenericElement provides a way for WebObjects to accommodate custom HTML elements that are empty. Since the HTML language is evolving rapidly, it's convenient to have a way to dynamically generate elements which are not explicitly supported by WebObjects.

In HTML, an *empty element* (for example <HR> or <BR>) is represented by a single tag and so can't enclose any text or graphics. In contrast, a *container element* (for example, <A ... > ... </A>) has opening and closing tags that delimit the text or graphic affected by the element. (See the related element WOGenericContainer for information about the support of container elements.)

elementName

Name of the HTML element to generate. **elementName** must be statically defined, that is, it must be a constant. It can't be something returned by a script method, for example. Please note that for elements with URL attributes, the URLs specified will appear as is in the HTML document.

This approach works for many elements, but has one limitation. Some HTML elements have an **href** attribute that associates the element with a URL. In WebObjects, the corresponding dynamic element generally has two mutually exclusive attributes, **href** and **action**, which make use of the HTML element's **href** attribute. (See WOHyperlink for an element that can have either an **href** or an **action** attribute.) The dynamic element's **href** attribute simply returns a URL, but **action** invokes a WebObjects method, which returns a URL. This overloading of the HTML **href** attribute is not supported by WOGenericElement. If your custom element requires this functionality, you will have to create your own subclass of WODynamicElement.

WOGenericContainer

Synopsis

WOGenericContainer { elementName = *aConstantString*; ... **};**

Description

WOGenericContainer provides a way for WebObjects to accommodate custom HTML container elements. Since the HTML language is evolving rapidly, it’s convenient to have a way to dynamically generate elements which are not explicitly supported by WebObjects.

In HTML, a *container element* (for example, <A ... > ... </A>) has opening and closing tags that delimit the text or graphic affected by the element. In contrast, an *empty element* (for example <HR> or <BR>) is represented by a single tag and so can’t enclose any text or graphics. (See the related element WOGenericElement for information about the support of empty elements.)

elementName
Name of the HTML element to generate.

This approach works for many elements, but has one limitation. Some HTML elements have an **href** attribute that associates the element with a URL. In WebObjects, the corresponding dynamic element generally has two mutually exclusive attributes, **href** and **action**, which make use of the HTML element’s **href** attribute. (See WOHyperlink for an element that can have either an **href** or an **action** attribute.) The dynamic element’s **href** attribute simply returns a URL, but **action** invokes a WebObjects method, which returns a URL. This overloading of the HTML **href** attribute is not supported by WOGenericContainer. If your custom element requires this functionality, you will have to create your own subclass of WODynamicElement.

WOFrame

Synopsis

WOFrame { value = *aMethod*; | **src** = *aURL*; ... **};**

Description

WOFrame represents itself as a dynamically generated Netscape Frame element.

value
: Specifies the method that will supply the content for this frame.

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

The **value** and **src** attributes are mutually exclusive.

Java Support: WOApplet and WOParam

Synopsis

WOApplet { code = *javaClassName*; ... **};**

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

Description

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.

code
: The name of the java class.

name
: The name of a parameter.

value
: The value of this parameter.

action
: The method that the applet will invoke.

WOEmbeddedObject

Synopsis

WOEmbeddedObject { value = *aMethod*; | **src** = *aURL*; ... **};**

Description

A WOEmbeddedObject provides support for Netscape plugÿins. 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 **value** attribute.

value

Specifies the method that will supply the content for this embedded object.

src

Specifies the external source that will supply the content for this embedded object.

The **value** and **src** attributes are mutually exclusive.