Dynamic Documents via Javascript

Matt Evett
Dept. Computer Science
Eastern Michigan Univ.

Introduction

- A *dynamic HTML document* is one whose tag attributes, tag contents, or element style properties can be changed after the document has been and is still being displayed by a browser
- We will discuss only W3C standard approaches
- Most of the examples use the DOM 0 event model so as to work with both IE6 and NS6
- To make changes in a document, a script must be able to address the elements of the document using the DOM addresses of those elements

Element Positioning & CSS-P

- HTML tables can be used for element positioning, but they lack flexibility and are slow to render
- CSS-P ("p" for "positioning")
  - CSS-P was released by W3C in 1997
  - CSS-P allows us to place any element anywhere on the display, and move it later
  - The position of any element can be dictated by the three style properties: position, left, and top
  - The three possible values of position are: absolute, relative, and static

Element Positioning (cont)

- ***Absolute Positioning***

```
<p style = "position: absolute;
left: 50px; top: 100px;">
```

- → SEE: absPos.html and Figure 6.1
- If an element is nested inside another element and is absolutely positioned, the *top* and *left* properties are relative to the enclosing element
- → SEE: absPos2.html and Figure 6.2

Relative Positioning

- *Relative Positioning*
  - If no top and left properties are specified, the element is placed exactly where it would have been placed if no position property were given
  - It can be moved later
- If *top* and *left* properties are given, the object is offset from where it would have placed without the position properties being specified
- If negative values are given for top and left, the displacement is upward and to the left
  - Can make superscripts and subscripts
- --> SHOW relPos.html & Figure 6.3. Note differences in vertical alignments

Static & Moveable Positioning

- Static positioning is the default, if *position* is not specified
  - Neither *top* nor *left* can be initially set, nor can they be changed later
- Moving Elements
  - If *position* is set to either *absolute* or *relative*, the element can be moved after it is displayed
  - Just change the top and left property values with a script
- --> SEE: mover.html & Figures 6.4 and 6.5

Element Visibility

- The visibility property of an element controls whether it is displayed
- The values are *visible* and *hidden*
- Ex: Suppose we want to toggle between hidden and visible, and the element's DOM address is dom

```
if (dom.visibility == "visible"
  dom.visibility = "hidden";
else
  dom.visibility = "visible";
```

- --> SHOW showHide.html
- Note that hidden elements are still allocated space.

6.5 Changing Colors and Fonts

- Background color is controlled by the `backgroundColor` property

- Foreground color is controlled by the `color` property

- Can use a function to change these two properties

- Let the user input colors through text buttons

- Have the text elements call the function with the element address (its name) and the new color

```
Background color:
  <input type = "text" size = "10"
         name = "background"
         onchange = "setColor('background',
                               this.value)">
```

- The actual parameter `this.value` works because at the time of the call, `this` is a reference to the

Changing Colors and Fonts

- Background color is controlled by the *backgroundColor* property
- Foreground color is controlled by the *color* property
- Can use a function to change these two properties
- Let the user input colors through text buttons
- Have the text elements call the function with the element address (its name) and the new color
- Background color:
- <input type = "text" size = "10" name = "background"
- onchange = "setColor('background', this.value)">
- The actual parameter *this.value* works because at the time of the call, *this* is a reference to the text box (the element in which the call is made)
  - So, this.value is the name of the new color
- → SHOW dynColors.html

Dynamic Colors and Fonts

- *Changing fonts*
  - We can change the font properties of a link by using the mouseover and mouseout events to trigger a script that makes the changes
  - In this case, we can assign the complete script to make the changes to the element's attribute (in the HTML):

```
onmouseover = "this.style.color = 'blue';
        this.style.font = 'italic 16pt Times';"
onmouseout = "this.style.color = 'black';
        this.style.font = 'normal 16pt Times';"
```

- SHOW dynLink.html

Dynamic Content

- The content of an HTML element is addressed with the *value* property of its associated javaScript object
- → SHOW dynValue.html

Stacking Elements

- The *top* and *left* properties determine the position of an element on the display screen, which is a two-dimensional device
- We can create the appearance of a third dimension by having overlapping elements, one of which covers the others (like windows)
- This is done with the *zIndex* property, which determines which element is in front and which are covered by the front element
- The stacking order can be changed dynamically

Stacking Example

- Make the elements anchors, so they respond to mouse clicking
  - The href attribute can be set to call a JavaScript function by assigning it the call, with 'JAVASCRIPT' attached to the call code:

```
<a href = "JAVASCRIPT:fun()">
```

- The handler function ("fun", here) must change the *zIndex* value of the element
  - A call to the function from an element sets the *zIndex* value of the new top element to 10 and the *zIndex* value of the old top element to 0
  - It also sets a variable (currentTop) to refernce to "top" element
- SHOW stacking.html

Locating the Mouse Cursor

- The coordinates of the element that causes an event are available in the *clientX* and *clientY* properties of the event object
  - These are relative to upper left corner of the browser display window
- *screenX* and *screenY* are relative to the upper left corner of the whole client screen
- - If we want to locate the mouse cursor when the mouse button is clicked, we can use the click event
- SEE where.html

Reacting to a Mouse Click

- A mouse click can be used to trigger an action, no matter where the mouse cursor is in the display
- Use event handlers for *onmousedown* and *onmouseup* for the document object to effect the action.
  - In the example, the action is to change the visibility attribute of a message
- SEE anywhere.html

Slow Movement of Elements

- To animate an element, it must be moved by small amounts, many times, in rapid succession
- JavaScript has two ways to do this, but we cover just one:

```
setTimeout("fun()", n)
```

  - fun() is called, then a delay of n milliseconds, then repeat the call

Slow Movement Example

- *Example*: move a text element from its initial position (100, 100) to a new position (300, 300)
- Use the *onload* attribute of the body element to initialize the position of the element (via its *top* and *left* attributes)
- Repeatedly call a function ("moveText") to change *top* and *left* by one pixel in the direction of the destination
- *A problem*: coordinate properties are stored as strings, which include the units ("150px")
  - So, to do addition or subtraction with the coordinate properties, we must convert them to just numbers; the units must be replaced before the properties are used
  - Use pattern matching to strip off the "px"

Possible Problems

- ***Another problem:*** We need to use some HTML special characters ('<' and '--')
  - We've avoided this problem before by placing these characters in html comments. But we might want our pages to be readable by XHTML parsers
  - XML parsers may remove all comments
  - Put the script in a CDATA section (but this wouldn't be readable by an HTML parser)
  - A solution: Put JavaScript in separate file, and reference it via the src attribute of the script element
- These are problems of validation only (the W3C html validator disallows these characters in XHTML docs)
- Both IE6 and NS6 deal correctly with commented HTML special-characters
- SHOW moveText.html

Dragging and Dropping

- We can use *mouseup*, *mousedown*, and *mousemove* events to grab, drag, and drop
- *Example*: magnetic poetry (dragNDrop.html)
  - Two static lines of text and a collection of "words" that the user can click & drag
- We use both DOM 0 and DOM 2 models:
  - DOM 0 to call the *mousedown* handler, "grabber"
  - The DOM 2 event model is required (the Event object and its property, *currentTarget*, to identify which "word" was clicked upon.)
- We use three functions: grabber, mover, and dropper

Drag & Drop, 1st Handler

- 1. Get a reference to the element to be moved, i.e. to the element under the cursor when the mouse button is pressed down (in the onmousedown handler)
- We can get the id of an element on which an event occurs with the srcElement property of an event object; srcElement has a property named id

```
event.srcElement.id
```

  - = the id of the element on which the event occurred
  - So we use this in the handler
  - The handler also registers handlers for mousemove and mouseup…

Drag & Drop, 2nd and 3rd Handlers

- 2. Move the element by changing its *top* and *left* properties as the mouse cursor is moved (onmousemove)
  - Use *event.x* and *event.y* to track the mouse cursor
- 3. Dropping the element when the mouse button is released by unregistering these two handlers.
- SEE dragNDrop.html