Adding Messages to a Picture
Using HTML and JavaScript to capture inputs and create text on canvas and hyperlinks

By Jeanine Meyer  jeanine.meyer@purchase.edu  http://faculty.purchase.edu/jeanine.meyer

HTML5 and JavaScript provide ways to capture input and add to the look and operation of a webpage. In this article, I describe an application in which the viewer specifies a font using radio buttons; sets the color, size and contents of the message; positions and orients the text using mouse actions; and provides a URL for a hyperlink. The article also demonstrates interactions with Cascading Style Sheets and dynamic creation and removal of HTML markup on the document.

Inspired by ideas of signing an image or simply adding a message, I came up with an application in which the site visitor is provided ways to place text on top of an image. I wanted the visitor to be able to specify the content and position and orientation of the text because I wanted to place text on a slant. I also wanted the ability to pick a font from a set of fonts, vary the size and choose a color.

Figure 1 shows the opening screen.

These screen shots are taken using the Chrome browser. Notice the blue block next to Pick color. This is one of the new HTML5 input types. If the viewer clicks on the block, a special panel appears providing the viewer a way to specify the color as shown in Figure 2. In certain browsers, Firefox for example, this feature degrades to being just a text field and users need to enter a 6 character hexadecimal code or one of the words accepted as color names.

Figure 2: Color pop-up panel invoked by the input type=color element

Figure 3 shows a screen show after the viewer has picked a color, adjusted the slider (input type='range'), entered the text I'm under the table into the Message text field and also typed in a website URL into the Website field.

The next piece of input is difficult to illustrate so I won't even try. The viewer positions the mouse cursor on the picture, presses down with the mouse button, drags and then releases. This defines a starting position and an angle.
The last step is to click on the **Add message** button. The program will use all the inputs, including the information generated by the mouse events, to draw the message on the screen and produce a hyperlink under the picture. Figure 4 shows the results.

![Figure 4: Result of adding a message](image)

Notice that the text has a shadow to improve legibility.

The text field under the label **Website** has more functionality than shown. Specifically, it is one of the new HTML5 input types, `type='url'`. If the viewer entered something that is not a standard URL, Figure 5 shows the result. The term for this is input **validation**.

![Figure 5: Error message displayed by Chrome when something not a URL was typed](image)

It is possible to place multiple messages, each with a new hyperlink. Figure 6 shows two added messages and links. Notice the different colors and fonts.

![Figure 6: Two messages, with associated hyperlinks](image)

The picture does not take up all the canvas, so this is why one of the messages appears in its entirety even though it is off of the picture. A **Clear messages** button is provided to remove all the messages and the links.

I will describe the individual features used in this application and then indicate how to put it all together. You can check out the hyperlink to the actual application given in the **Learn More** section.

### Radio buttons

Radio buttons are a set of buttons in which only one can be set. They are specified in HTML as `<input>` elements of `type='radio'` and with a common name. Look ahead to Listing 4. The input elements are in a form named “f”. The three radio buttons all have the same setting for name, `name='font'`. This will produce an array that can be accessed using `document.f.font`. You will see that next to each input element, I coded a span element. This produced three, visible ABCs, each in a unique font.

The challenge in this application is to set the font for the text to be drawn on canvas to be the font that the viewer indicates when clicking one of the radio buttons. Look ahead now to the complete definition of the function `addmessage` in Listing 6. The first step is to determine which radio button was clicked. My code performs this operation using a for-loop that cycles through the `document.f.font` array in the usual way. The length of the array is `document.f.font.length`. The expression `document.f.font[i].checked` is true if the ith radio button is checked and `document.f.font[i].value` returns the corresponding value.

The next step is to use this value to determine the actual font. I will describe that in the next section. By the way, if the viewer does not click on a radio button, my code has set a default value.

### Fonts, picking up style information, and creating a shadow

The best practices recommended for HTML documents is to put formatting information, such as specification of fonts, in the **style** element; implement dynamic behavior using JavaScript in the...
script element; and put content and structure in the body element. For this reason, I didn't want to list the set of fonts in JavaScript in the script element as well as in style directives, even though my list consisted of just 3 fonts. Instead I wanted the JavaScript to reach into the CSS so to speak to get the specific font family.

Before continuing, let me remind you that the recommendation for specifying fonts is to specify multiple fonts just in case your first choice is not available. Listing 1 shows the style element for this application. This sets of formatting for elements with ids equal to font1, font2, and font3.

```
<style>
#font1 {font-family:"Comic Sans MS", cursive;}
#font2 {font-family:"Broadway", Arial, Helvetica, sans-serif;}
#font3 {font-family:"Lucida Calligraphy", Times, serif;}
</style>
```

Listing 1: The style element

The span elements that I defined next to the radio input elements has the ids "font1", "font2", and "font3", respectively. The value of the clicked (checked) radio button is 1 or 2 or 3. My code sets a variable called `elemname` to be "font"+document.f.font[i].value and then uses that value to set a variable `elem` to be document.getElementById(elemname).

```
function getff(elem){
  // IE
  if (elem.currentStyle) {
    return elem.currentStyle["fontfamily"];  
  // other browsers
  } else if (document.defaultView &
    document.defaultView.getComputedStyle) {
    return document.defaultView.getComputedStyle(elem, null).getPropertyValue("font-family");
  // fallback
  } else {
    return null;
  }
}
```

Listing 2: The getff function

Prompted by my daughter, who told me it was a common technique, I decided to put a drop shadow around the text. HTML5 JavaScript provides 4 different attributes of the canvas context that are relevant for this effect. The variable `ctx` holds the canvas context, so the amount of horizontal offset is set using `ctx.shadowOffsetX`, vertical offset using `ctx.shadowOffsetY`, the amount of blurring, `ctx.shadowBlur` and the shadow color, `ctx.shadowColor`. Drop shadow settings apply to things other than text, so since I redraw the image on the canvas, I need to reset these attributes to the original settings immediately after drawing the text.

Color input type

The figures, especially Figure 2, have shown you the utility of the color input type provided by HTML5. The HTML markup I used was:

```
<input name="color" type="color" value="#0000ff" />
```

The value is specified using the 6 character hexadecimal format, also indicated as RRGGBB since the first two hexadecimal characters indicate a red setting going from 0 to 255, the next two a green setting and the last a blue setting. For my initial value, I

![Figure 7: Opening screen using the Firefox browser](image1)

The viewer can enter hexadecimal values or choose from among the set of recognized names. Figure 8 shows the results of typing in teal.

![Figure 8: Screen shot using Firefox](image2)

Notice also that I neglected to type in an actual website, but the program accepted http://. It would not have accepted something that did not start with http.

Once the color is set, either by user action or inaction in which case the original value is preserved, the statement

```
ctx.fillStyle = document.f.color.value;
```

sets up the color for any subsequent drawing, which in this application will be a call on `ctx.fillText`. 
Slider (range input type)

The slider is implemented with the HTML5 input type="range". This also provides a way for programmers to set minimum and maximum values. My code for the slider is in the form element in the body element:

```html
<input name="size" type="range" min="10" max="70" value="30"/>
```

What is termed graceful degradation of the range input type to a text field in Firefox does not preserve the check on minimum or maximum values.

I use this to set size in the `addmessage` function. First, my code sets a variable

```javascript
sizeff = document.f.size.value;
```

Then I use this variable in two places to combine with a font specification to set a variable I named `fontfamily`.

```javascript
fontfamily = sizeff+"px Comic Arial"; //default
```

and

```javascript
fontfamily = sizeff*"px *"getff(elem);
```

Later, I use this to set `ctx.font`.

There is an inconsistency in implementation here: the default value of the size is set in the `<input name="size" ... > element but not in the set of radio buttons for the fonts. It is possible to set one button as initially checked, but I chose not to do that.

Mouse events and angle calculation

My application requires a starting position for the text message and an angle. I set global variables `mx = 350 and my = 400` for the default position, and `mangle=Math.PI/6` for the angle. Remember that JavaScript uses radians as opposed to degrees for angles.

Clockwise is positive. So this code means the initial angle, the default if the viewer does not do anything with the mouse, is 30 degrees counter-clockwise.

The plan is to use the two mouse positions to define a starting position for the text written on the canvas and an angle. Event handling is set up in the `init` function (see Listing 5) using `addEventListener` citing the function `markstart` for the `mousedown` event and the function `markend` for the `mouseup` event. The name `markend` does not signify the end of the text message, but remember that this code does give the desired results. Check it out.

```javascript
function markstart(ev) {
    var fx;
    var fy;
    if ( ev.layerX || ev.layerX == 0) { // Firefox, Chrome
        fx= ev.layerX;
        fy = ev.layerY;
    }
    else if (ev.offsetX || ev.offsetX == 0) { // Opera,
        fx = ev.offsetX;
        fy = ev.offsetY;
    }
    else {
        return; //no ev event
    }
    mangle = Math.atan2(fy-sy,fx-sx);
}
```

Listing 3: The mouse event handlers

As you can detect in the code, I use trigonometry in the `markend` function to compute the angle. The `atan2` function uses vertical change and horizontal change to compute an angle.

**Note:** the attentive reader may ask, “What about the fact that the coordinate system for the canvas is upside-down, with higher vertical values occurring lower down on the screen?” It turns out that this code does give the desired results. Check it out.

Now, the actual drawing of the text on the screen is done using methods that transform the coordinate system of the canvas. Look ahead, again, to Listing 6, the `addmessage` function. My code uses a `translate` operation to move the coordinate system to the starting point, `(mx, my)` and at that point, does a `rotate` transformation of the indicated angle. Because I want my code to draw more messages, I need to write code to undo these transformations. This is done using `ctx.save()` before the transformations, and `ctx.restore()` afterwards.

URL input type and generation of HTML markup

The URL input type specifies input as being a valid web address.

```html
<input id="site" name="site" type="url" value="http://... ">
```

Note that the initial value I wrote in the input field is acceptable (see Figure 8), because it starts with `http://`, even though it is a nonsense value. In the `addmessage` function, the statement

```javascript
dest= document.f.site.value;
```

sets the variable `dest` to be either the initial value I wrote or the value entered by the viewer. The `addmessage` function creates an element dynamically. The code sets the value of the `href` attribute and the visible contents of the hyperlink.

Creating HTML markup dynamically is done using the `createElement` method of the document. So the statement

```javascript
created = document.createElement("div");
```

creates an element. The element now needs some content, which is provided using
Now the element exists and has the appropriate content and just needs to be attached to the HTML document. I also want to remember these elements, so to speak, so I, or rather my code, can remove them later. The appendChild method attaches the element and the push method adds the element to an array I named createdlist.

createdlist.push(document.body.appendChild(created));

The clearmsgs function, described in the next section, will use the array to remove the hyperlink elements.

With this background on the individual parts of the application, it is time to put it all together.

Putting it all together

The application consists of the usual style, script and body sections. The body is shown in Listing 4. A table serves to format the screen with the canvas, where the picture is drawn in one column and the form in another.

Listing 4: The body element of the application

The JavaScript makes use of certain global variables and an init function, invoked by action of the onload attribute in the body section. Listing 5 holds the code declaring, and in most cases assigning, initial values to the global variables, and the definition of the init function.

Listing 5: Global variables and init function

The previous sections described how the individual form input values are referenced to create the text message. The bulk of the work is done by the addmessage function, so I will show it next. The addmessage function is invoked by action of the onsubmit setting in the form. Information is picked up from the form elements and the values as calculated by the mouse event handlers or the original values of mx, my and mangle, are used to draw the text. Listing 5 shows the code.

Listing 6: The addmessage function

The clearmsgs function clears the canvas and then redraws the image. Next a for-loop is used to iterate through the createdlist array and invoke the removeChild method to remove the created HTML markup. Finally, the createdlist array is reset to be an empty array. The function is shown in Listing 7.
Listing 7: The clearmessage function

```
function clearmsgs() {
    ctx.clearRect(0,0,800,600);
    ctx.drawImage(pic,0,0,757,505);
    for (var i=0; i<createdlist.length; i++) {
        document.body.removeChild(createdlist[i]);
    }
    createdlist = [];
}
```

**Warning:** the intended function of the different input types is to perform checking (validation) so we do not have to do it in our JavaScript code. However, given the uneven implementation, it is best to still do the check. My example DOES NOT do this for the URL input or the color or the size.

A function table is a good way to provide an overview and serve as documentation for an application. Table 1 is a function table for this application.

<table>
<thead>
<tr>
<th>Function</th>
<th>Invoked by</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>init</td>
<td>Setting of onLoad in <code>&lt;body&gt;</code></td>
<td></td>
</tr>
<tr>
<td>markstart</td>
<td>addEventListener in init</td>
<td></td>
</tr>
<tr>
<td>markend</td>
<td>addEventListener in init</td>
<td></td>
</tr>
<tr>
<td>addmsg</td>
<td>Setting of onSubmit in form</td>
<td></td>
</tr>
<tr>
<td>getff</td>
<td>Invoked in addmsg</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Functions

**Other input types to explore**

HTML5 does provide other new input types, including `tel`, `date`, `time`, `week`, and `email`, and some, though not all browsers provide some level of checking. If the browser does not support the type, then it generates an ordinary text field.

The `datalist` feature is a way of specifying a list of options for input. The options are not visible but provide what is termed an auto-complete facility. I chose to use the `radio` buttons in conjunction with the `span` elements because I did not want names of fonts, but, instead, visible ABC strings to display using the indicated fonts.

You can use this application as a way to provide your viewers/users/customers/clients the facility to mark up a picture and add favorite links, or serve as a model for useful features of HTML5 and JavaScript. Enjoy.

**Learn more**

There are many sources, online and in-print and some sort of e-books, for learning HTML5 and JavaScript techniques. Here are links to my recent books and the website for this example.

- **The Essential Guide to HTML5: Using Games to learn HTML5 and JavaScript**, [http://www.friendsofed.com/book.html?isbn=9781430233831](http://www.friendsofed.com/book.html?isbn=9781430233831). This is a text for beginners at programming as well as more experienced programmers who want to learn about HTML and JavaScript, including the new features of HTML5, such as video.

- **HTML5 and JavaScript Projects**, [http://www.apress.com/9781430240327](http://www.apress.com/9781430240327). This book is more advanced than the first one. It contains several applications involving video, including bouncing video, combining media with Google Maps for a quiz, and a jigsaw turning into a video clip.

- To see the final version of the application and to view all the source code, go to [http://faculty.purchase.edu/jeanine.meyer/html5/addmessage.html](http://faculty.purchase.edu/jeanine.meyer/html5/addmessage.html).

Jeanine Meyer lives just north of New York City and currently teaches at Purchase College/SUNY after many years at IBM, doing research on robotics and manufacturing and consulting on educational grants. She likes providing programming examples for her Mathematics/Computer Science and New Media students and colleagues and really, really likes working with images and video clips of her granddaughter and other family members.