

Upcoming Standards Related to Graphics, Data Storage and Script Processing in Future Web Applications

Shwetank Dixit
Opera Software ASA,
Waldemar Thranes Gate 98, Oslo, Norway
shwetankd@opera.com

Abstract— HTML5 is one the verge of making web applications more powerful and easier to code. However, there are other W3C standards which are also, in their own way, trying to make web applications more powerful, aesthetically better and easier to code for. Hence it makes sense to take a look at how all these standards could be used by developers, and what it could mean for the future of the web. This paper will cover other upcoming standards such as CSS3, W3C Web Storage, Web SQL DB, Web Workers API, Canvas 2D API and others and see how these could work with HTML5 and its features in order to make the next generation of web applications.

Keywords— CSS3, HTML5, Web Storage, Offline Applications, Web Storage, Canvas

I. INTRODUCTION

HTML5[1] is one the verge of making web applications more powerful and easier to code. However, there are other W3C standards which are also, in their own way, trying to make web applications more powerful, aesthetically better and easier to code for. We will look into some of the W3C Specifications and see what they are and how they can effect the future of web applications.

II. ADVANCEMENTS IN STYLING OF WEB PAGES USING CSS3

CSS3[2] is the further advancement of the Cascading Style Sheet specification. The work on CSS3 is modularised into different parts. These parts will be separate recommendations in themselves.

A. CSS3 Web Fonts

CSS3 Web Fonts[3] provide a way to use custom fonts on web pages. Till now, user agents only supported six font types for web pages across platforms. CSS3 Web Fonts provides a way to display text in a web page using any font in .ttf or .otf format. The advantages of this approach include better aesthetic matching of the text to the general design of pages, and also better readability in certain scenarios, especially in the case of non-latin fonts.

CSS3 Web Fonts are introduced using an '@font-face' property. An example of it would be

```
@font-face {src: url('sampleFont.ttf');font-family: 'myFont';}
```

And then using it in the same way as a normal font, such as

```
#samplediv{font: 2.5em 'myFont';}
```

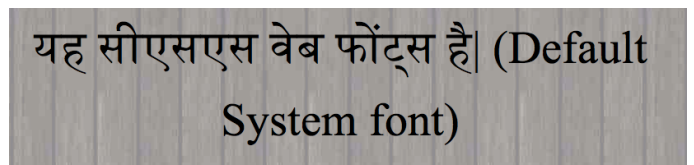


Fig. 1 Example of Hindi text in web page using the default system font available on the Mac OSX platform

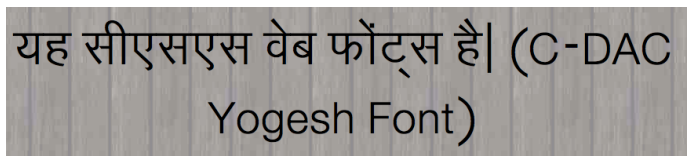


Fig. 2 Example of Hindi text in a web page using C-DAC Yogesh Font, using CSS3 Web Fonts.

As mentioned earlier as well, this can be especially useful in presenting non-latin based languages in a more readable format. We shall take the example of Hindi.

Figure 1 shows hindi text in a web page using the default system font available. Figure 2 shows the hindi text using C-DAC Yogesh font (not available on the system by default) and embedded in the page using the @font-face property of CSS3 Web Fonts.

B. Rounded Corners with CSS3 'border-radius' Property

'border-radius'[4] is part of the CSS Backgrounds and Borders Module Level 3 Candidate Recommendation[5] by the W3C. A part of a page, usually an <div> element, having rounded corners, is a very common effect used by developers. However, so far, it has been done usually by using images and using multiple nested <div> tags. Not only does it unnecessarily increase the complexity of the code, but also consumes greater bandwidth because of the images being used.

>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum

Fig. 3 Example of div element in a web page using rounded corners generated through CSS3 border-radius property.

It works as:

border-radius: p, q, r, s;

Where p , q , r and s are the top-right, bottom-right, bottom-left and top-left border radii respectively. Many browser vendors currently support this property through their own specific vendor prefixes (-o, -moz and -webkit prefixes), however, the eventual aim is to move away from vendor prefixes and have developers use one standard 'border-radius' property.

C. Opacity and Precision Colour Control

CSS Color Module Level 3[6] provides the ability to control the opacity[7] of elements as well as have precise control over their colour.

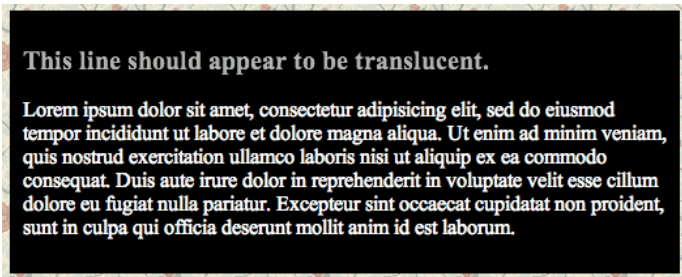


Fig. 4 Example of an <H2> header element in a web page appear translucent using the 'Opacity' property.

It works as:

opacity: x

Where x is a numerical value between 0.0 and 1.0.

Colour control is improved by providing two mechanism, namely, RGBA[8] and HSLA[9]. RGB corresponds to the Red-Blue-Green colour model and HSL corresponds to the Hue-Saturation-Lightness colour model. the 'a' in both corresponds to the alpha channel which can give added opacity control to the colours.

This box should appear to be translucent.

>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Fig. 5 Example of div element in a web page appear translucent and consequently a partial view of the background image being present.

In both RGBA and HSLA, values can be defined either as integers, or as percentages. However, both cannot be used in combination to define a value.

An example of RGBA and HSLA would be:

color: rgba(100%, 50%, 20%, 0.9);

color: hsla(80%, 70%, 60%, 0.5);

The ability to add such colour controls makes it possible for designers to have a precise control over their designs.

III. WEB STORAGE AND OFFLINE APPLICATIONS

A. W3C Web Storage API

The W3C Web Storage API[10] is designed to be a better way to store data on the client side. It makes it possible to store key-value pairs on the client side. It has two parts - Local Storage and Session Storage.

1) *Session Storage*: Session: Storage[11] is storage meant only for that particular session. As soon as the browser tab or window is closed, all data stored in Session Storage is supposed to be lost. This is useful for applications such as shopping sites etc where highly sensitive transactions take place, which should not be reproduced by opening the tab again

2) *Local Storage*: Local Storage[12] is meant to be a persistent form of storage. It is meant to store value persistently, which means that if the browser tab or window is closed, but re-opened (or if the user goes to that same page again), the information should still be available.

Key-Value pairs are stored in Session and Local Storage as:
 sessionStorage.setItem(yourkey, yourvalue);
 and
 localStorage.setItem(yourkey, yourvalue);

Data is retrieved for session storage, by:
 var data = sessionStorage.getItem(yourkey);
 and for local storage by:

```
var data = localStorage.getItem(yourkey);
```

The specification also mentions a storage event to be fired whenever there is change to the storage present. With it, it is possible to know the nature of the storage, the previous and new value of the storage, the URL of the page whose key has changed, as well as the key itself.

It is also interesting to note that there is another specification, called the W3C Web SQL Database Specification[13], which specifies how to store, retrieve and manipulate data on the client side using SQL like queries. However, as of now, that specification is in a state of impasse, as there has not been a clear consensus on how to support the SQL dialect, which has been left as a reference to that of SQLite's, so far.

B. HTML5 Offline Applications

The HTML5 specification also deals with how applications could work offline. The specification outlines something known as 'application cache'[14]. Under this, the files needed for the application to run offline are referenced in a file called a 'manifest file'. This manifest file can tell the browser which files to cache for offline use. These files can be anything which is required to run it, i.e, images, pages, JavaScript files, videos, CSS files, etc.

Much of the application cache API is non-normative, however, it has laid out certain constants to describe the current state of the application cache ('idle', 'checking', 'downloading', etc), which are returned when invoking 'cache.status', which tells the status of the application cache.

Application Cache (especially, if combined with a persistent client side data storage mechanism like Web Storage or Web SQL DB) can be very useful in providing the web application offline, when the user does not have connection to the internet. In particular, mobile devices can benefit from it, as in many cases, signal might be low or nil, but the user might want to still access the web application.

IV. OTHER STANDARDS AND PROPOSALS

A. Web Workers

Web Workers is[15] a draft by the W3C which outlines how JavaScript, usually relatively expensive JavaScript in terms of computation and/or time, could be done on the background, in different 'workers', in a thread-like way. In other words, it provides a mechanism for performing background scripts without interrupting other scripts such as those for clicks and other user interaction elements.

It is possible to create a worker to run from a script, which could be an external file. Also, it is possible to create a shared worker, so that requests from multiple files could be handled.

A new worker can be created as follows:
`new Worker("worker.js")`

Where 'worker.js' contains the script needed to be executed as a worker. However, stipulations include the fact that workers do not have access to the DOM (Document Object Model) of the page, and neither do they have direct access to the parent page[16]. The worker is supposed to be strictly for computation, and not designed for user interactive behaviour. Communication between the worker and the page is done using the `postMessage` API[17] which part of HTML5 Web Messaging[18]. The ability given by Web Workers to process code in the background could be very useful in complicated web application which require heavy JavaScript computation.

B. Canvas 2D API

The `<canvas>` tag[19] is part of the HTML5 specification. However, the API required to make graphics with it have been moved to a separate specification, called the Canvas 2D API [20]. The `<canvas>` tag is basically similar to an empty `` element, except in this, you can program the pixels using scripting. The API to do this, is specified in this API. This makes it possible to draw programmable 2D graphics, by making use of this API and JavaScript.

To draw using canvas, the first thing to do is to get the context using the `getContext` method

For example:
`context = canvas . getContext(contextId)`

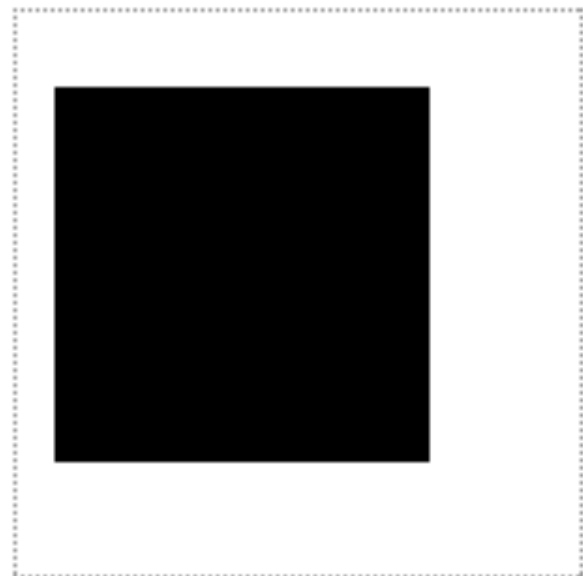


Fig. 6 Example of black square drawn within a canvas element with a dotted outline.

Basic shapes like ellipses and rectangles (which also means, in effect, circles and squares) can be made, as well as arcs, lines and curves. The API provides ability to draw quadratic curves as well as bezier curves.

For example, a simple rectangle can be coded by writing:
`context.fillRect(0,0,50,100)`

Besides drawing of shapes, various effects such as transforms, rotating, scaling and translating are also provided in the API.



Fig. 6 Example of gradient drawn with the canvas element and API.

Specifying colours, alpha channels, gradients and patterns are also available. 2D Graphics made with the help of this API can be made interactive and action driven, by combining them to work with user driven events, such as on 'onclick' event handler.

V. CONCLUSIONS

Future web application would in all likelihood be using HTML5 in combination with other upcoming standards and

technologies. These standards have their own use specific and unique uses, providing the ability for future web application to have graphics in a better and easier fashion, better readability for many languages on the web, for storage of data on the user's side and for offline web application, as well as for processing of intensive JavaScript methods concurrently in a thread-like manner.

REFERENCES

- [1] *HTML5 Working Editor's Draft*, W3C, 2010.
- [2] *Introduction to CSS3: Working Draft*, W3C, 2001.
- [3] CSS Fonts Module Level 3 Available: <http://www.w3.org/TR/css3-fonts/>
- [4] The 'border-radius' Property. Available: <http://www.w3.org/TR/css3-background/#the-border-radius>
- [5] *CSS Backgrounds and Borders Module Level 3: W3C Candidate Recommendation*, W3C, 2009
- [6] *CSS Color Module Level 3: W3C Working Draft*, W3C, 2008
- [7] Transparency: the 'opacity' property. Available: <http://www.w3.org/TR/css3-color/#transparency>
- [8] RGBA Color Values. Available: <http://www.w3.org/TR/css3-color/#rgba-color>
- [9] HSLA Color Values. Available: <http://www.w3.org/TR/css3-color/#hsla-color>
- [10] *Web Storage: W3C Editor's Draft*, W3C, 2010
- [11] The Session Storage Attribute. Available: <http://dev.w3.org/html5/webstorage/#the-sessionstorage-attribute>
- [12] The Local Storage Attribute. Available: <http://dev.w3.org/html5/webstorage/#the-localstorage-attribute>
- [13] *Web SQL Database: W3C Editor's Draft*. W3C, 2010
- [14] Application Caches. Available: <http://dev.w3.org/html5/spec/Overview.html#appcache>
- [15] *Web Workers: W3C Editor's Draft*. W3C, 2010
- [16] Computing with JavaScript Web Workers. Available: <http://dev.w3.org/html5/spec/Overview.html#web-workers>
- [17] The postMessage function. Available: <http://dev.w3.org/html5/postmsg/#dom-window-postmessage>
- [18] *HTML5 Web Messaging*, W3C, 2010
- [19] The Canvas Element. Available: <http://dev.w3.org/html5/spec/Overview.html#the-canvas-element>
- [20] *Canvas 2D API Specification 1.0*, W3C, 2009