Mary Nyamor

Publishing Multimedia on the Web

Manipulating video in real-time

Helsinki Metropolia University of Applied Sciences
Bachelor's Thesis
Bachelor of Media Engineering
Thesis
Date: 5 December 2011

Author(s) Title Number of Pages Date	Mary Nyamor Publishing multimedia on the web 76 pages + 2 appendices 5 December 2011
Degree	Bachelor of Engineering
Degree Programme	Bachelor of Media Engineering
Specialisation option	Printing and Multi-Channel Publishing
Instructor(s)	Pyry Ahlfors, Project Manager Kari Aaltonen, Principal Lecturer

The purpose of this thesis was to explore the different ways of publishing multimedia content on the web using HTML5 to allow an easier and more effective implementation of the dynamic content on the web.

A brief history and overview of HTML and its progression are covered to show how HTML5's development came about. HTML5 features are discussed focusing mainly on HTML5 video, audio and canvas. The practical part of the project includes the development of a custom player that is implemented using HTML5, JavaScript and cascading style sheets.

With the help of this thesis, one would be able to build a custom video player that would allow easier customization with JavaScript and cascading style sheets, depending on the developer's competence. It would also allow the developer to develop the application further as the basics are provided in this thesis. The results of the project were tested on different devices and browsers in order to see how they were supported. This provided us with excellent results that allowed us to stream clips across different browsers considering it was a demo application. It is however important to keep in mind that the standards of HTML 5 are still under development and may therefore be changed at any time.

Keywords	HTML5, DOM, video, JavaScript, CSS3, CSS, Canvas, XHTML, HTML4, API, RIA, multimedia, web applications

Contents

Abbreviations and Terms

API	Application Programming Interface, a convention for accessing the functionality provided by a program module.
RFC	Request for Comments, a numbered memorandum published by the IETF
CSS	Cascading Style Sheets, the style sheet language of the Web.
HTML	HyperText Markup Language
RIA	Rich Internet Applications
UX	User Experience
SGML	Standard Generalized Markup Language
WHATWG	Web Hypertext Application Technology Working Group
DTD	Document Type Definition
PDF	Portable Document Format
MIME	Multipurpose Internet Mail Extensions
DOM	Document Object Model
SVG	Scalable Vector Graphics
SMIL	Synchronized Multimedia Integration Language

1 Introduction

The goal of this bachelor's thesis is to provide a custom HTML5 video player that works in today's web browsers. This would aid in creating a solid foundation that would allow video publishers who, with the rapid growth of mobile web, face a fragmented environment in which they can no longer reach their desired audience using flash player alone. The solution will provide web users with an optimal video experience across web browsers and portable devices such as ipad. Additionally, it tries to define a common ground, if one exists, and define how far standardization has come.

This project was done for a company called ZAAZ. It is a full service interactive agency that focuses on strategy, design, development, UX, web analytics, optimization, social networking and search marketing. They were in need of a video solution that would deliver content accordingly by helping them achieve a seamless support of both flash and HTML5. The introduction of HTML5 has made this possible due to the introduction of the <video> tag whereby developers no longer have to rely on Adobe Flash Media player to play H.264 video as a cross platform solution. However, since the most recent version is still very much under development, the implementations of the standard are expected to vary across browsers. A test project was therefore done to find out if the current specifications were sufficient enough to stream videos across different browsers and devices.

This study gives developers possible ways on where to start when they want to use existing API'S for their video content streaming. It also focuses on helping developers get a clear understanding of the background and importance of multimedia on the web. Flash having been the most used Internet standard for video delivery in the previous years, has allowed this new markup to pave way and it is now possible to stream videos not only generically but also natively. Additionally, with Google having made a decision early this year to remove H.264 from Chrome, it will be beneficial for publishers and developers when it comes to selecting the right video format for online distribution.

2 Background of multimedia content on the web

"Every time your web browser requests a page, the web server sends a number of headers before it sends' the actual page markup" [22]. These headers are important, because they tell your browser how to interpret markup and render a particular resource. An example of how this works can be seen in figure 1 below. Early web servers did not transmit the Content-Type header, because they did not exist. However, on the webpage's, scripts, multimedia content and anything with a URL was served to the user with a specific MIME type in the Content-Type header. In order to understand the history of HTML5 and the motivations behind it, the need to understand the technicalities specifically, Multipurpose Internet Mail Extensions (MIME) type is necessary [22].

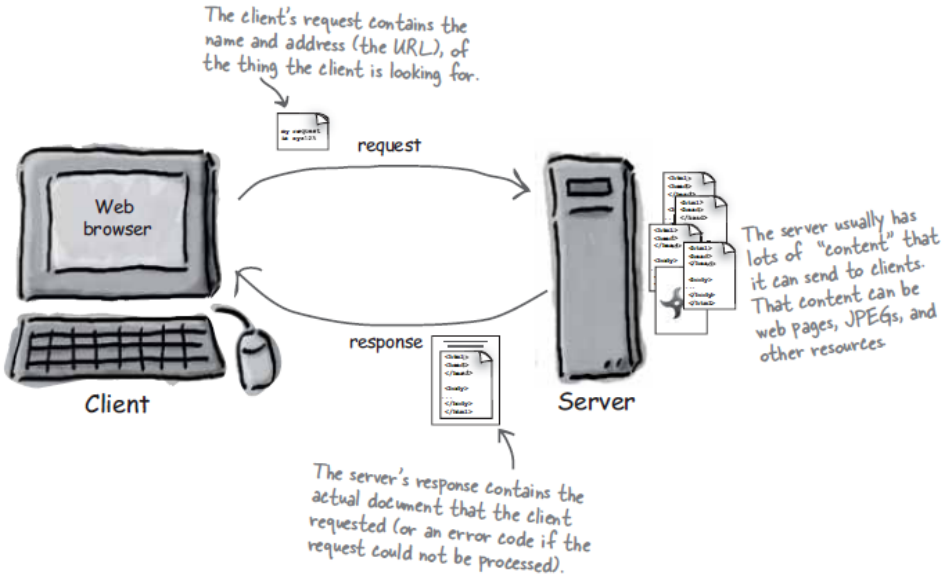

Figure 1. Client - server communication.

This chapter reviews a brief history of Hypertext Markup Language (HTML), which, in principle, is the basic building block of webpage's that uses semantic markup language to describe webpage's. That is, it emphasizes the meaning of the encoded information over its presentation. It goes on and further gives a general overview of what is new, what is the same and what has been removed from the previous versions that make it HTML5. The chapter will focus more on the new markup since it builds upon the previous versions and its aim is not to explain in depth what HTML is but rather to give an understanding of the new HTML5 multimedia.

2.1 Brief history of web trends

Web history dates back in the early 90's when a man called Tim Berners-Lee invented the web with HTML as its publishing language [4]. This was based on Standard Generalized Mark-up (SGML), an internationally recognized method for marking up texts. A year later, Tim Berners-Lee released the first version of his browser [4]. During that time, the language was very limiting and therefore developers could only write simple text to the web. It was not long before Dave Ragget, from Hewlett-Packards Labs, published a draft specification for HTML+, which included the IMG tag that allowed not only text but also images to be included on the web pages [5]. Figure 2 below displays different browsers rendering content in early 90's.

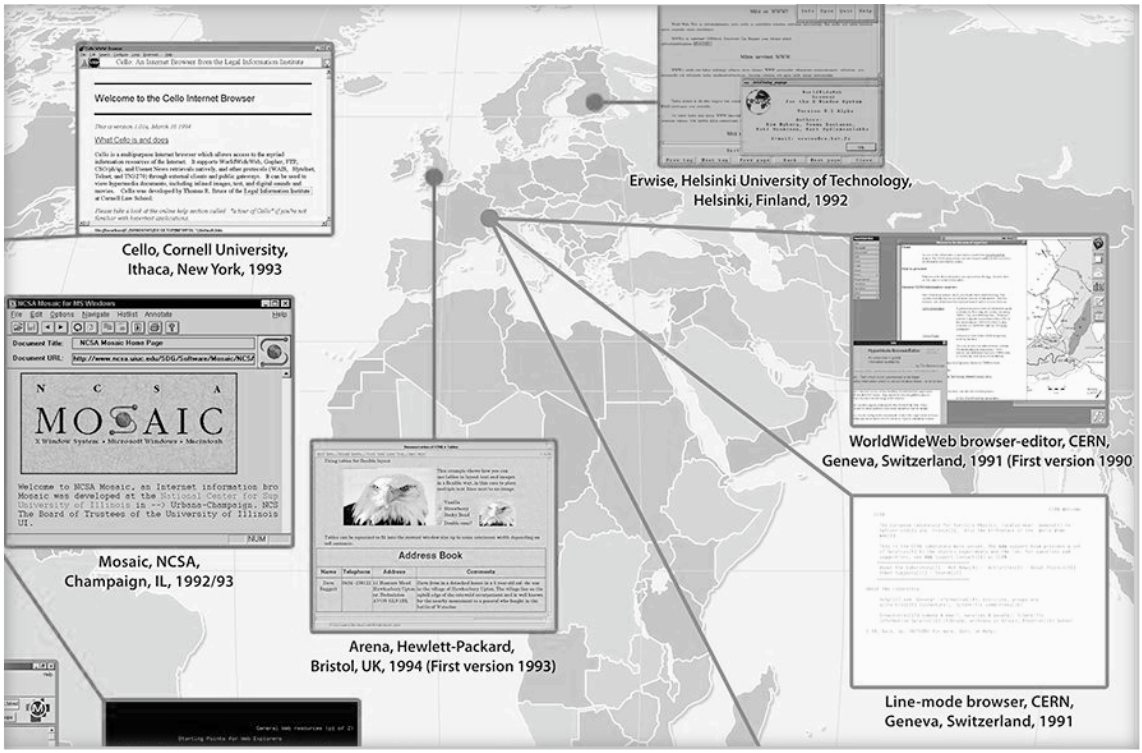

Figure 2. Browsers rendering text and images

Attempts to make some additional improvements on the web later followed and this led to the publication of a new version, HTML 2.0. This included ideas from former versions and also included a Document Type Definition (DTD) but was of no use to browsers. Later, it became RFC proposed standard and was seen as the base standard by which all browsers were measured until HTML 3.2. The specification stated that: -

> HTML is an application of ISO Standard 8879:1986 Information Processing Text and Office Systems; Standard Generalized Markup Language (SGML). [25]

Its properties defined the basic performance of almost all current browsers. The momentum picked up quickly and soon there was a need to enhance the look and feel of websites. Soon HTML 3.0 was released and thus allowed the use of cascading style sheets to be adopted [6]. This led to the introduction of the STYLE element and the CLASS attribute, which lived on in HTML 4 [4].

Trouble later started when Netscape decided to introduce new tags and attributes into their browsers that forced developers to replicate tags and this led to confusion [4]. This version did not match what was being implemented in browsers and was therefore abandoned [4]. Microsoft later released Internet Explorer as the interest of the Internet was expanding. Through the next few years, W3C published the specification for HTML 3.2 [4], [7]. The specification stated: -

> HTML 3.2 aims to capture recommended practice as of early '96 and as such to be used as a replacement for HTML 2.0 [25].

Relative to version 2.0, Added features such as text flow around images; tables, applets, superscripts and subscripts were widely deployed [25]. This aided in improvement of multimedia content display on web pages.

As the standards were evolving pretty steadily, a new version, HTML 4.0 was published as a W3C recommendation [4]. This allowed better development of websites as more additional features were introduced. Some of these features included more multimedia options, scripting languages, better printing facilities, style sheets and documents that were more accessible to users with disabilities. It was then revised without incrementing the version number and adopted the model proposed in the experimental RFC on HTML tables dropping a few presentational attributes [29], [9], [8].

Around the late 90's, the document was again revised and named HTML 4.0.1 [10]. Improvements could be seen and developers were now able to enjoy a wider range or target media. They could markup a test description of an included object, better tables that included captions, column groups and mechanisms that facilitated non-visual rendering, better forms and better distinction between document structure and presenta-

tion. Multimedia features allowed users to play videos and audio and even download them [30]. This became one of the successful versions of HTML after a long period of versioning. It has since been used and is still being used by developers. Support across browsers has been good so far, [8], [9], [10].

At the end of the same year, an interim specification that simply reformulated HTML in XML without adding any new elements or attributes was drafted. This specification became known as XHTML 1.0, which stands for Extensible HyperText Markup Language [27]. This was a stricter and cleaner version of the markup. It defined a new MIME type for XHTML documents, application/xhtml+xml. Beginning the following year, XHTML became a W3C recommendation. Later in May 2001, the first edition of XHTML 1.1 was published, which added a few minor features on top of the previous version.

Figure 3 below shows the times when these transitions occurred.

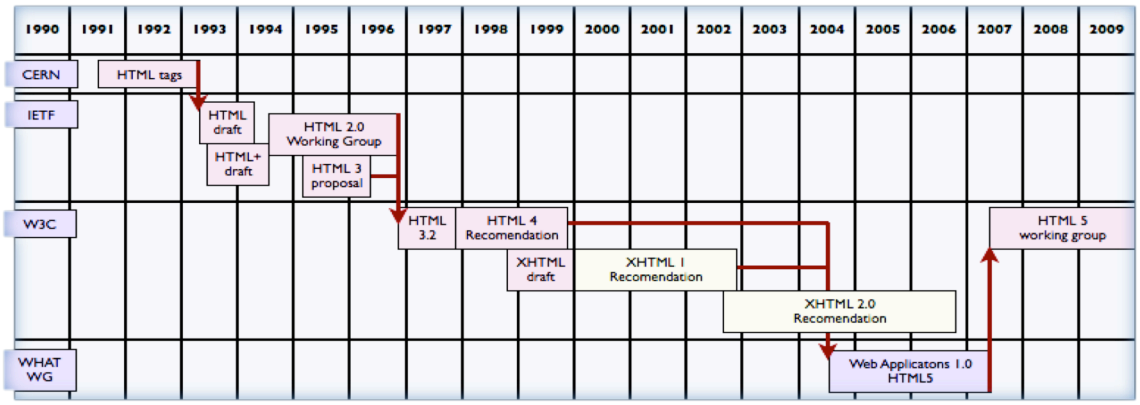

Figure 3. Web trends

As seen from Figure 3 above, the world's first public web page went online in 1991 at CERN, the European Laboratory for Practicle Physics in Geneva Switzerland [1] and a later copy of it was archived by the World Wide Web Consortium (W3C). Since then, it has cultivated many web-oriented standards such as the HTML, XHTML, SVG and PNG file formats. Starting in mid-2000, another version was launched by W3C group. Their purpose was to do away with this markup and introduce the world to a whole new markup language called XHTML2, which was XML, based. Since the idea was a disaster, a group independent of W3C called the Web HyperText Application Technology

Working Group (WHATWG) continued evolving HTML with a focus on web applications [13]. Later in the late 2000's Mozilla, Opera and Apple requested that the W3C adopted the work under the HTML working group as HTML5. It is now the current and latest version.

The term HTML5 denotes the abstract hypertext markup language that can be written in both HTML and XHTML syntax, as described by its specification [31]. Although it's not a web standard yet, latest versions of web browsers have started to implement essential parts of HTML5. It defines HTML syntax that is compatible with HTML4 and XHTML1 documents [33]. It is however not compatible with the obscure SGML features of HTML4, such as processing instructions and shorthand markup as these are not supported by most user agents. Its development aims to allow better error handling, reduce need for external plug-in, have more markup that will replace scripting and introduce new features based on CSS, DOM, JavaScript and HTML.

When intending to code using this new markup, the first thing you notice is that the new markup has its DOCTYPE simplified compared to the traditional HTML. An example of this can be seen below.

Figure 4. Differences between traditional DOCTYPE and the latest DOCTYPE

HTML5 specification has also included a series of new semantic elements that used to give meaning to various parts of a web page. The new features that have been introduced include the canvas element that is used for drawing, video and audio for media

playback, new content specific elements such as header, nav, footer, article and section; better support for local storage and new form controls like date, calendar, time, url, email and search. All these features aid in developing web application with the use of individual API'S. It should be noted that HTML5 is currently not an official standard and therefore no browser supports it fully.

2.2 Current state of web trends

In the rapidly growing online industry, web technologies are constantly developing forcing content and application providers to constantly add new layers of functionality to their products. This has resulted to the birth of new techniques and applications. HTML, for instance, was primarily created to statically display images and text on web pages and not designed for high interactivity. Today, however, a clear trend can be seen where more frameworks for creating rich user experiences appear [39]. Moreover, it has transitioned into a widely and highly visible platform for delivering rich media experiences.

As previously mentioned, multimedia content was first published when the <img> tag was introduced. Integration of such content, in particular motion picture, followed the same way, but much later. In the absence of a standard way to include clips on a webpage, video publishers have mostly been dependent on third-party plug-ins such as Apple QuickTime, Adobe Flash Player or Silverlight. Only when Synchronized Multimedia Integration Language (SMIL) introduced multimedia elements could web documents really include continuous media. SMIL was followed in this direction by Scalable Vector Graphics (SVG) and more recently by HTML5.

Rich Internet Applications (RIA) such as Java, Adobe Flash, Silverlight and Adobe Flex have been used to host multimedia content and have in turn represented the transition in development of web applications. For a long time, they have provided richness and interactivity that lacked in the previous web applications. As with any new technology, they have provided opportunities as well as challenges. As web users needs grow in complexity, their need to incorporate rich interactive features of web applications in a simpler and more effective way arises. Challenges such as lack of a common standard, accessibility and security concerns have forced web trends to continue evolving. This

has in turn led to evolvement of HTML leading to us having a more interactive, simpler and richer HTML version known as HTML5.

Today we talk about Web Applications and Rich Internet Applications (RIA), which have naturally used proprietary technologies because they offer new possibilities [38]. A paradigm shift could be happening now that open web standards are beginning to gain the lead that other technologies have had. For instance, multimedia and interactive graphic capabilities are now added to the mix making HTML5 the main topic for many developers and designers today. With the new improvements, we are able to see that web applications are in a sense easy to upgrade since they come from a central platform unlike installed computer programs. It is however, not yet possible to make web applications look and behave like installed programs due to disputes to do with graphics, performance, client side storage and geo-positioning [33].

2.3 HTML5 as a Rich Internet Application

Over the years, rich Internet application platforms have been adopted since they have been seen to manifest clear benefits such as improved customer satisfaction and reduced application maintenance costs. As technologies like AJAX and JavaScript libraries flourish, the potential of HTML and JavaScript have become more apparent. As a result, HTML 5 has promised to provide all the benefits of RIA by building upon open standards instead of using these third party technologies [36]. Continued advancement of these web standards has enabled web developers to build almost anything with HTML, CSS and JavaScript. With HTML5 still under development, new features such as canvas, video and audio elements have already allowed easier rendering of web content by making it possible for developers to include motion pictures on web pages with a single line of code and without the need of a plug-in based solutions such as Adobe flash, Microsoft Silverlight or Sun JavaFX.

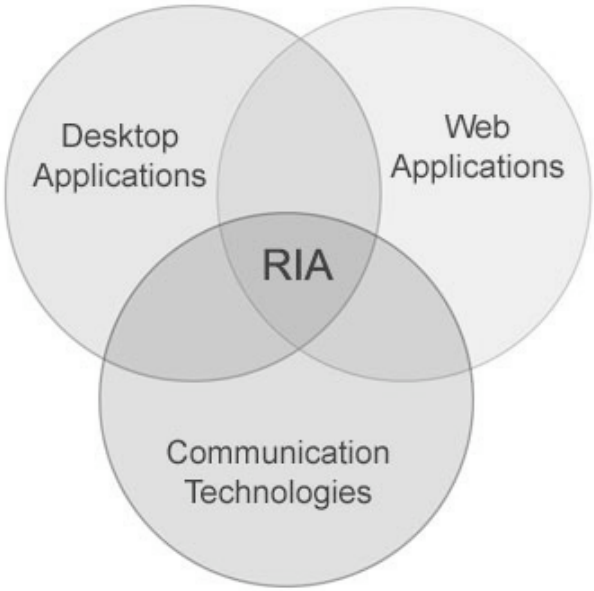

Figure 5. Rich Internet Applications

As can be seen from the above figure, there's no specific outline that can clearly identify and determine what is a Rich Internet Application and what is not. While it's possible that in the long run, HTML 5 will become an acceptable alternative of some types of RIA, it's not yet there [37]. This is mainly because it's still under development thus the specification is deemed to change at any time. Meaning, concerns such as connectivity, user experience and web response time are still a major challenge as improvements are being made to solve them. Before it becomes a ratified standard, browser vendors will rely on partial support as the current expectation of when the standard will be fully approved as a W3C standard will be about ten years from now. In the long run, it will have a significant impact on how Web applications are built; but as a complementary technology to leading RIA platforms, not a replacement [35], [36].

Currently HTML5 is seen to be one of Adobe's competitor as it posses a threat to the company. This is due to the fact that it tries to minimize or do away with the use of proprietary technologies such as Flash; which has for a long time been the main source of streaming multimedia content. Lucky for Adobe, HTML5 is still under development and is currently only supporting the latest browsers hence still needing help from flash to stream content in older browser versions. In the near future, however, the state of publishing multimedia content on the web might be possible, as users will no longer have to rely on these technologies.

3 HTML5 Multimedia in depth

Publishing of video on the web has become fairly simple since the introduction of HTML5. Motion pictures and other forms of multimedia are quickly governing the user experience by adding value to all businesses simply because users are reading less and watching more. This is mainly because web design and development has gone beyond traditional definition. With new trends evolving, HTML5 has allowed code minimalism using few elements as possible to achieve an easier to read and less congested code whilst still providing full functional software. In addition, content providers such as YouTube are now able to seamlessly integrate HTML5 content and even allow video manipulation using JavaScript and cascading style sheets thus allowing consistency and interactivity around web pages [20].

Web users are now able to stream clips from anywhere thanks to continual increase of bandwidth and technologies for multimedia content. So far, many browsers have integrated HTML5 support in order to break the restrictions imposed by proprietary browser plug-ins. Its development has allowed open technologies to deliver high quality user experience to stream multimedia content. Its introduction has also enabled iphone and ipad users to stream videos since these devices lack flash support. At the same time, it has enabled browser stability as plug-ins can cause instability and create worry to some users when prompted to download and install newer versions.

HTML 5 aims to move the web away from proprietary technologies such as plug-in based RIA frameworks. It is currently competing with Flex, Silverlight and JavaFX, and could make such plug-ins appear unnecessary. It will however take time for HTML 5 to get equipped and penetrate the market. At the time of writing, experts at both Microsoft and Adobe thought that a major breakthrough lies five to ten years into the future [19], [39].

This chapter focuses more on the multimedia elements in particular, the video element. It goes on and further talks about its composition, some of the available formats as well as browser support. The chapter will also touch on the audio element but will not go deeper on it.

3.1 Anatomy of the video element

In recent years, developers in need of embedding clips on web pages have been using the <object > element. The use of this element, which is a generic container for "foreign objects," also required the need to use the previously unfounded <embed> element and duplicate many parameters due to browser inconsistencies. This resulted in code that looked much like this: [14].

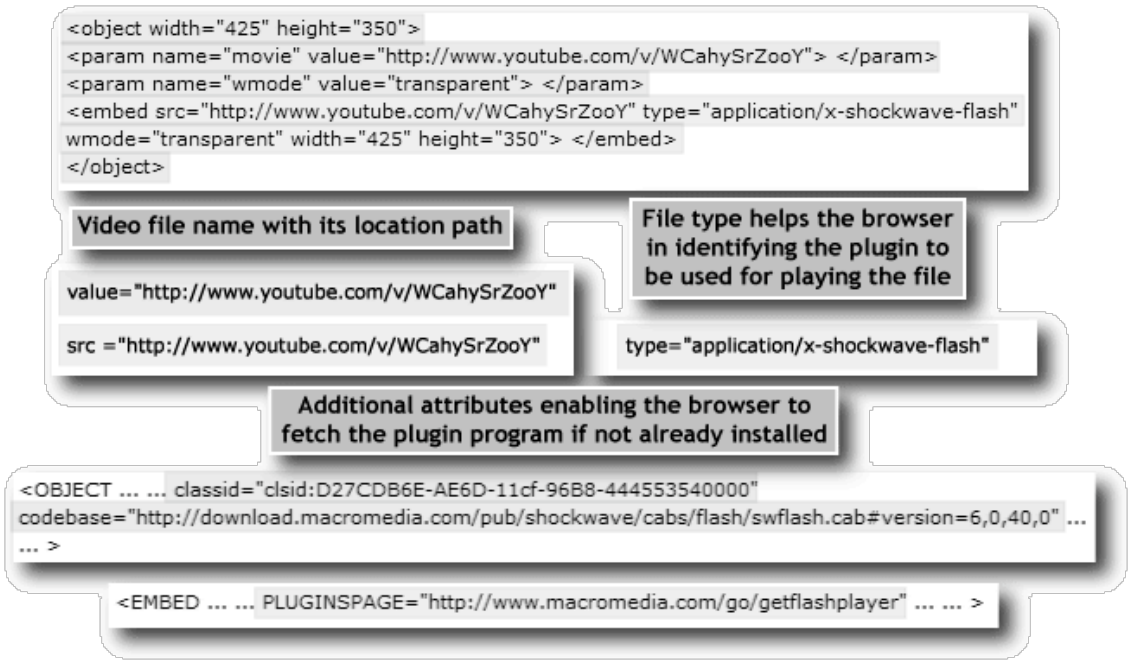

Figure 6. Object Embed code

Today however, developers are able to include motion pictures on the web with a simple line of code thanks to the new markup. HTML5 specification has defined a standard way of including them on web pages through the introduction of a new element called the <video> element. This element enables browsers to natively playback video without the need to install any additional plug-ins like Flash, Silverlight, or QuickTime. In addition, a set of standard JavaScript APIs has been provided to allow developers to create their own playback controls, should they wish to do so [15]. An example of the new video code is shown below.

```
<video src="video.ogg" width="200" height="200"></video>
```

Figure 7. Simple video code

From the figure above, the .ogg file extension is used to an Ogg Theora video while the src indicated is an attribute that defines its URL or path. A key issue with this element, however, is that the file formats supported by each browser are dependent on patents whereby some browsers have restriction to other patents. More of this will be discussed later in the chapter.

When talking about anatomy of this element, we are basically talking about its composition. Inside the element, there are several attributes, each with a specific function. These attributes allow the video to perform certain functions. These attributes can be grouped either as display attributes such as src and controls or they can be grouped as playback attributes such as currentTime, duration, paused, loop, seeking and ended. These attributes can further be grouped as read-only for example duration and ended while other such as currentTime and volume can be read and written. [21]

Each of the attributes mentioned has a value that aid in giving a detailed description to the video regarding its performance. Accessing of these attributes is done either through inclusion in the video tag as shown in figure 7 or through JavaScript. An example of how these attributes can be made to function using JavaScript is shown in figure 8.

```
var videoObject = document.getElementById("video");
    videoObject.muted = false;
    videoObject.currentTime = 0.0;
        if(videoObject.ended){...}
```

Figure 8. Using JavaScript to perform an event

What the code is basically doing here is that it sets the initial value of mute to false before the video starts playing. That means, the video will have sound and will only go on mute once the mute button is clicked. More attributes that can be included in the video tag are discussed briefly explained in table 1 below.

Table 1. Video attributes

Attribute	Usage
Autoplay	Allows the video to automatically start playing once loaded. It is a Boolean value that is either true or false to allow video playback automatically.
Controls	Allows the developer to set a control bar below the video with the play, pause and stop buttons. This is a Boolean attribute.
Height	Sets the height value of the video.
Loop	Can either true or false and this in turn states if the video plays over and over or not.
Buffered	An object of ranges which are already buffered
Preload	Controls whether the video is preloaded into the element usually taking three values; auto – lets browser decide if it preloads, none – no preloading required and metadata which tells the browser that detecting metadata of the video is a good idea.
Poster	Holds the URL of an image if the video is unavailable.
Src	Holds the URL or path of the video.
Width	Sets the width of the video.

In addition to these attributes, the element also can include more information such as buffering, playback time and error messages. A detailed code showing the video element with different attributes can be seen below.

```
<video width="320px" height="240px" autobuffer="autobuffer" autoplay="autoplay"
       loop="loop" controls="controls" poster="/_img/videoPoster.jpg">
     <source src='video.mp4' type='video/mp4; codecs="amp4v.20.8, mp4a.40.2"'>
     <source src='video.ogv' type='video/ogg; codecs="theora, vorbis"'>
</video>
```

As seen from the code above, different formats can be included in the source element, which currently supports around four different formats with flash included.

3.2 Anatomy of the audio element

The audio element is meant for playing sounds or streaming audio files using the <audio> tag. It also uses a single line of code to include an audio file on a webpage and has limited formats that play across browsers. Formats that are supported include MP3, WAV and Ogg Vorbis. The following code shows how audio can be inserted on a webpage.

```
<audio src="mymusic.ogg" controls="controls"></audio>
```

Like the video element, the audio element also has its attributes that play a similar role. The <audio> shares a lot of markup and functionality with the <video> element, but lacks the @poster, @width, and @height attributes, since the native representation of an <audio> element is to not display visually [21]. An example of how audio attributes can be placed within an element is shown below.

```
<audio controls="controls">
        <source src="mymusic.ogg" type="audio/ogg">
        <source src="mymusic.mp3" type="audio/mpeg">
</audio>
```

From the code shown above, it can be seen that the source of the audio files has been included twice. This has been done so that in case one of the audio formats is not supported, the browser will check to see if the other format is supported. The browser will then play the suitable format. Like the video, there is no single encoding format that will be supported by all the web browsers. Some of the attributes used for the audio element are shown in the table below.

Table 2. Audio attributes

Attribute	Usage
@src	Used to include the audio on the webpage as it fetches the audio file from a specific location or path where the audio file has been stored.
@autoplay	Allows the audio file to start playing automatically as soon as the browser has downloaded and decoded sufficient audio data
@loop	This is a Boolean attribute. It allows the audio file to play over and over again making the audio restart each time the playback is complete.
@controls	Used for user interactions. This allows the user to have full control of the audio.

As shown above, it can be seen that these attributes are not so different from the video attributes. They perform the same function and have the same names. The only difference comes from the source file, whereby we find that it holds different formats. Like the video, these attributes can also be controlled through the use of JavaScript. An example of an audio component created using JavaScript and cascading style sheets is shown below.

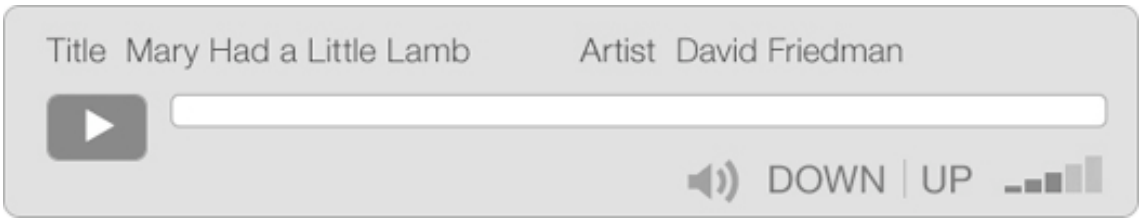

Figure 9. An example of an audio player

The figure above is an example of an audio player, which consists of different audio controls that aid in playing the loaded audio file. The audio track and artist name has also been added to make the audio player more interactive.

3.3 Video codecs and containers

Playing a video file entails different things. These include, format interpretation as well as decoding of the audio and video stream. This video file can be said to be a con-

tainer, which is similar to a zip archive that contains a number of files. [17]. These files within are known as streams or tracks and represent a variety of things including video, audio and subtitles. According to specifications, there is no specific container or codec you have to use. One can therefore specify multiple clips and the browser will play using the first file it identifies. This is because different browsers support different containers and codecs and there is not one browser that fully supports all these formats. Some examples of containers include: - Flash video (.flv), Audio Video Interleave (.avi), MPEG 4(.mp4) and Ogg (.ogv).

A codec on the other hand is an algorithm that encodes the video stream. They are used to encode or decode a particular stream to allow playback. In practice, it would be when your player decodes the stream depending on the codec, which then displays a series of images and frames on the screen [22]. Browsers that support the video element have codecs installed by default, which allows easier playback. As supported in the book of Bruce and Remy (2011), it can be seen that: -

> Early drafts of the HTML5 specification mandated that all browsers should at least have built-in support for multimedia in two codecs: Ogg Vorbis for audio and Ogg Theora for movies [14].

These codec's were abolished after Nokia and Apple objected hence leading to having no defined codec in the specifications [14]. At the moment, not many formats are supported. Some of these video formats currently in use include: -

- MPEG4 (with H.264 video codec and AAC audio codec)
- OGG (with Theora video codec and Vorbis audio codec)
- WebM – VP8 +Vorbis (Google's WebM technology, which is currently supported in Opera and will be supported in upcoming versions of the Firefox and Chrome browsers

Firefox, Opera, and Google Chrome have selected the Ogg/Theora codec. Firefox 4 also supports the WebM codec. Safari and Internet Explorer support the MPEG4/H.264 codec. Vorbis audio codec and the Theora video codec are freely available and can be embedded on any container, while the use of the MPEG-4 and H.264 codecs are pat-

ent-encumbered as they are subject to license fees [22]. More about browser support is discussed later in the chapter.

H.264 also known as MPEG-4 Advanced Video Coding was developed by the MPEG-4 group and standardized in 2003 as a successor to earlier standards such as MPEG-2. It is currently one of the widely used formats for video compression, recording and distribution of high definition video. It aims to provide single codec for use in high definition systems as well as low-resolution devices such as mobile phones. H.264 delivers better visual quality at smaller file sizes, takes longer to encode and require more CPU power to decode in real-time as compared to MPEG-2 and MPEG-4 (DIVX or XVID). [22]

Theora, on the other hand, evolved from VP3 codec and has subsequently been developed by Xiph.org Foundation [22]. It is a free, open-standard container format supported natively by Firefox 3.5+, Google Chrome, Opera and most distributions of Linux. One can also watch Ogg content on Mac OS X and windows by installing appropriate codecs. It is a royalty-free codec. Theora video can be embedded within any container format although it is mostly seen in an Ogg container [3]. Within Ogv container we usually have the audio tracks known as Theora and video tracks known as Vorbis [3].

Google introduced the WebM video format in May 2010 as a high quality format that is free for both implementers and publishers. WebM files have the .webm extension and consist of VP8 video and Ogg Vorbis audio in a container based on Matroska. WebM represents a significant development in the codec landscape. As far as browsers go, at least Firefox, Opera, and Chrome will support WebM natively. Opera 10.6 is already shipping with WebM support. Mozilla and Google have committed to shipping WebM in the next versions of their browsers [22]. Like Theora, WebM is also a royalty-free codec and can be substituted as .ogv.

Including motion pictures using the <video> tag requires provision of different format. This is because no single combination of the codec's works in all browsers. For compatibility reasons it is best if all three versions are linked to the element and a fallback is provided. This helps the users to be able to play the video in one or the other platform.

When comparing the quality between the common H.264 and WebM, they both seem to have the same quality though in H.264 is of better quality. The difference is minimal such that consumers are most likely not to notice. In order for the web developer to include all the codecs, separate <source> elements instead of src are used for each encoding with appropriate type attributes inside the <video> element, which lets the browser download the format that it can display [14]. The example below shows how this is done.

```
<video controls>
    <source src='video.mp4' type='video/mp4; codecs="amp4v.20.8, mp4a.40.2"'>
    <source src='video.mpg' type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
    <source src='video.ogv' type='video/ogg; codecs="theora, vorbis"'>
</video>
```

The code above shows that it is possible to include many files in the video element using the source element. This allows extensive browser coverage. From our example, the first file that is supported by the browser will be rendered. If the browser does not support the file and attribute is not specified, it should attempt to render the other two files or try retrieving the media from the server. In addition to the src, the video codec is also included to the linked source file as it allows the browser to make smart decision.

3.4 Browser support

Support across browsers requires more than just inclusion of the video or audio tag on a webpage. In addition to using multiple source tags for codec specification, fallback options and JavaScript come handy when thinking of for better browser coverage across different platforms. This allows users visiting your site to play videos regardless of the platform they are using. Table 3 and 4 below shows how video and audio formats are supported.

Table 3. Browser support for HTML5 video

Codec/Container	IE	Firefox	Safari	Opera	Chrome	iPhone	Android
Theora+Vorbis +ogg	-	3.5+	-	10.5+	5.0+	-	-
H.264+AAC+MP4	9	-	3.0+	-	5.0+	3.0	2.0+
WebM	-	4.0+	-	10.6+	6.0+	-	2.3+

As seen from table 3 and table 4, none of the browsers fully supports all the formats. It can also be seen that only the later browsers are supported. Internet explorer is not supported but maybe newer version such as version 9 could support the video tag.

Table 4. Browser support for HTML5 audio

Browser	Wave	Ogg Vorbis	Mp3
Safari	YES	NO	YES
Chrome	YES	YES	YES
Opera	YES	YES	NO
Firefox	YES	YES	NO
IE	NO	NO	YES

Since not so many formats are supported, the need to convert the videos in the correct format is necessary should the web publisher lack the needed format. It is however, recommended that one should convert the video from its source format instead of using a compressed version as this reduces the quality of the video. There are selective choices of video converters. Miro video converter is one example that can be used for both Macintosh and windows that allows conversion of H.264 or Ogg Theora for different devices [22]. VLC is also another converter that allows conversion but this is only available for windows and linux users. In addition to these, users are also able to convert their videos on various Internet archives. In the meantime, however, videos are still widely streamed through third party plug-in such as flash. However, as the situation currently stands,

> Any video publisher that wants to create web pages with videos that are expected to universally work with any browser will be required to publish video in at least two formats: in MP4 H.264/AAC and in either Ogg Theora or WebM [21, 7].

A flash fallback will still be required in order to satisfy all the users. This works in a way such that, if the video were not HTML5 compatible, it would revert to the third party plug-in. A fallback option for using the source tag helps users be able to play the video across different browsers. It is usually so that when using flash or Silverlight as your fallback, H.264 is enough. The example below displays a fallback action.

```
<!DOCTYPE HTML>
<html>
  <body>
    <video width="1280" height="720" controls="controls" poster="imt.jpg">
      <source src="imt.mp4" type="video/mp4" /> <!— H.264 -->
      <source src="imt.ogg" type="video/ogg" /> <!— Ogg -->
      <embed src="imt.swf" type="application/x-shockwave-flash"
      width="1080" height="720" allowscriptaccess="always"
      allowfullscreen="true">
      </embed>
    </video>
  </body>
</html>
```

Figure 10. Video code with flash fallback

The above code tries to show how a developer can include a flash as a fallback patio for browsers that do not support html5 video. Our example shows that HTML5 video is first supported; if not, then the "imit.swf" flash option is used. This allows users who lack support to still view the video in other versions. In addition to flash, other fallback options that can be used include the use of the Java Applet, which is useful for browsers that lack Java support and also don't support HTML5 video.

Our .ogv file extension is used to point to the ogg video. When all fails, a video download link can also be provided for old browsers that do not support the native video as this enables one to watch the video later from his or her operating system. WebM format can also be included as part of the source or as alternative to the two mentioned sources in the figure above.

4 Canvas and Styling

A canvas is a rectangle or a two-dimensional grid on a page on which you can use JavaScript to manipulate or draw things like shapes or display graphics [22]. In other words, HTML5 canvas element can be said to provide an API for rendering graphics or visuals. The canvas can better be seen to be described in detail according to Peter Lubbers, et al. (2010), where they state that: -

> The canvas concept was originally introduced by Apple to be used in Mac OS X WebKit to create dashboard widgets. Before the arrival of canvas, you could only use drawing APIs in a browser through plugins such as Adobe plugins for Flash and Scalable Vector Graphics (SVG), Vector Markup Language (VML) only in Internet Explorer, or other clever JavaScript hacks [17].

It supports the two dimensional drawing operations used across modern frameworks. Canvas element is simply created by using the code shown below

```
<canvas height-”200” width=400” id=”myCanvas”> </canvas>
```

This code is relatively straightforward. You simply need to specify the width and height of the canvas element and an ID for referencing it in the JavaScript. The width and the height indicated are just for example purposes. It should however be known that the size of the canvas greatly affects how fast an image is drawn on the canvas. So if you need to draw on a larger surface, one should use scalable vector graphics (SVG), which usually deals with vector shapes [32]. But since we are dealing with image manipulation, canvas is the better option between the two as it’s easy to implement and it does not store every objects for every primitive it draws [17] .In order to get the graphics displaying, JavaScript will be needed.

The element support for browsers is fairly good. However, since the semantic tag is still new, it should be noted that full support across browsers would not happen in the immediate future. A substitute for browsers that do not support this element such as some of the IE versions can be supported by the use of the Silverlight library called HTML5 canvas or use of excanvas which translates the API to Microsoft VML. Browsers

such as Firefox and Webkit have different judgements regarding the opening and closing tags and whether the canvas element should have separate closing tag between the opening and closing tags or not. For browsers that do not support the canvas, JavaScript can be used to display an error message. When it comes to using the canvas on the video element, the drawImage() function is used to achieve this.

In my application, you will be able to see how canvas can be used to manipulate video pixels. This will allow us to see different possibilities the canvas is able to offer and come up with awesome effects. The Chroma key example uses the canvas element that allows us to see how video can be edited online.

4.1 Drawing using the canvas

As mentioned earlier, the canvas element is like a painting canvas that allows you to draw or paint on it. This allows implementation of image processing through manipulation of pixels. An example of a canvas script being accessed through the document object model (DOM) is shown below.

```
<script>
        window.onload =function()
        {
        var canvas= document.getElemetById("canvas");
        context = canvas.getContext ("2d");
        Context.fillRect=(10, 20, 200,100);
        }
</script>
```

From the script shown above, the window.onload is our onload handler which helps us draw after the page is fully loaded. We then create a reference to the canvas using the getElementById to get it from the document object model and create a 2 dimensional context in order for us to draw. Once this is done, we use the 2D context and draw a filled rectangle on the canvas. The first two numbers, 10 and 20 indicate the x and y

positions of the canvas whilst the last two 200 and 100 give the width and height in pixels. To retrieve the context, the canvas uses the Document Object Method, getContex, which has only one parameter, the type of context. The result from this will be a rectangle with black fill. This is shown in the figure below.

Figure 11. An example of a simple canvas

When colour is not specified, the result will always have a black fill. Whenever we need to change the colour, we can use the fillStyle attribute and this is usually placed before we call the context.fill rectangle. This can be done like this.

```
var canvas= document.getElemetById("canvas");
  context = canvas.getContext ("2d");
  context.fillStyle= ('red');
  Context.fillRect=(...
```

The resulting colour will be red instead of black. Other styles can also be used whenever we want to draw something. One advantage of the canvas over Scalable Vector Graphics is that once you have created your canvas, you can save it as a bitmap image. Other properties such as fillRect, strokeRect and clearRect can be used in place of fillStyle.

Coordinates on canvas are always determined using pixels, the origin usually being at point "0,0" and the pixel values usually increase as you move along the x and y-axes. When you want to manipulate pixels on canvas, the getImageData is used [11]. It

usually represents the pixel data of the current state of the canvas context. Pixel data is the pixel array containing the color components of each pixel of the image data having the RGBA (red, green, blue and alpha) values each from 0 to 255. In this kind of manipulation, there are three specific methods used to perform the operations required [16]. These include: -

- createImageData – Takes two parameters (width and height) and creates transparent black pixels.
- putDataImage - It takes three parameters: the imageData and the x and y coordinates that are used when you need to modify pixels and draw them back on the canvas context. For example: context.putImageData(imagedata, 0, 0);
- getImageData – Takes four arguments that allow you to take a block of pixels on a specific region and examine them. For example: context.getImageData(x, y, width, height);

A representation of the pixels would look something like this.

←--------- Pixel I -------→ ←--------- Pixel 2 -------→ ←--------- Pixel 3 -------→

R	G	B	A	R	G	B	A	R	G	B	A
0	1	2	3	4	5	6	7	8	9	10	11

←---------- -Array Index- ----------→

Table 5. Representation of pixels

```
var canvas= document.getElemetById("canvas");
    context = canvas.getContext ("2d");
    var imagedata=context.createImageData (canvas.width, canvas.height);
    var canvasPixelArray = imageData.data;
```

As you can see from our representation, we call the createImageData the method takes two arguments, which include the width and height. These will then return the CanvasPixelArray that contains the RGBA color values. After this, you can now manipulate the pixels as much as you like by looping through every pixel on the canvas and changing its color values [16]. For example

```
for (var i=0;i< pixel.length;i+=4)
{
var red =pixel[i];        // red channel
var green= pixel[i+1]; // green channel
var blue= pixel[i+2];  // blue channel
var alpha=pixel [i+3]; // alpha channel
}
```

After this you can invert the color using the color range from 0 to 255.

```
pixel[i] = 255 -red;        // red channel
green= pixel[i+1] = 255 -green; // green channel
blue= pixel[i+2] = 255 -blue;  // blue channel
```

The result from this is an inverted image, which uses the putImageData that allows the placement of the image data in the same format as it was from the start. This then updates the image to reflect the new pixel values. An example of this can be seen below.

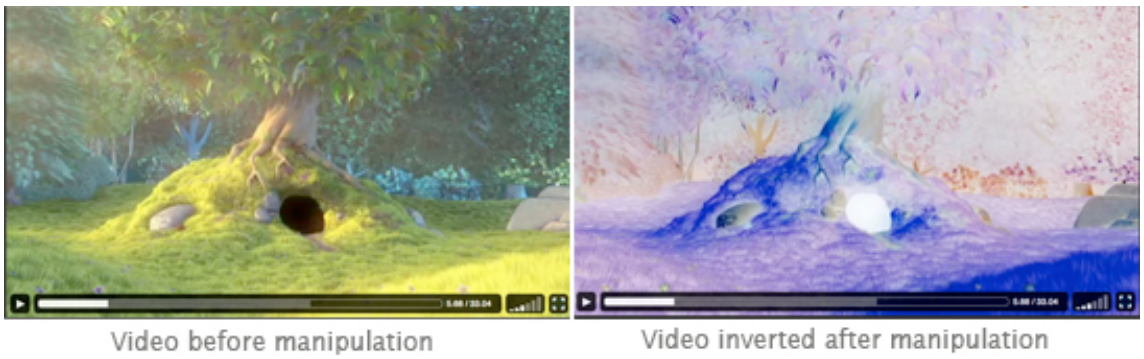

Figure 12. Pixel manipulation using the canvas

These video pixels above are manipulated using the ImageData object, which, as mentioned previously, consists of width, height and the data: which is the pixel array.

This process works in a way that allows the image data values representing each pixel to be calculated. These pixels are seen to be located at any given coordinate (x, y) whereby the component values are located. An example of how the pixel mapping process works is illustrated below [14].

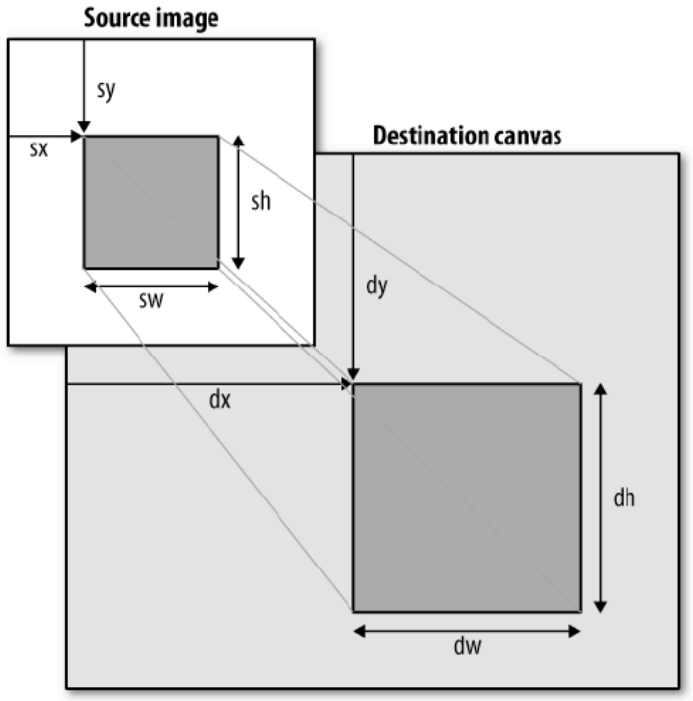

Figure 13. Mapping of a canvas using the drawImage() function

From the above picture, the image is first loaded then the drawImage method is called which takes the three arguments; image, x and y coordinates for the 2d rendering context. This is then placed inside the image load event [11]. This then creates the image on the canvas.

4.2 Canvas browser support

Knowledge of browser support is useful as it helps a developer know whether his application will be supported or not. All major browsers that support HTML5 support the canvas element. Like the video tag, a fallback text can be used for browsers that do not support canvas. It is however advised that, when adding a fallback, going dynamic is a better option as sometimes, just adding text with the words "your browser doesn't support canvas" or whatever words one uses might fail and give you the text even if

the browser supports it. This is because sometimes even after matching your canvas, it might not be accurate hence cause failure to load.

As mentioned before, the use of excanvas script can also be used for Internet explorer. However,

> Explorer Canvas initializes its own fauxcanvas interface automatically whenever you include the excanvas.js script in your HTML page. But that doesn't mean that Internet Explorer is ready to use it immediately. In certain situations, you can run into a race condition where the faux-canvas interface is almost, but not quite, ready to use. The primary symptom of this state is that Internet Explorer will complain, "Object doesn't support this property or method" whenever you try to do anything with a <canvas> element, such as get its drawing context [22].

The only solution for this would be to defer all manipulations until the onload event fires. This is because if you have multimedia elements that need to be loaded, they will first need to load then once they have finished loading, then the script will perform its magic. The table below shows how the canvas is supported across browsers.

Table 6. Browser support for canvas

Blackberry	IE	Firefox	Safari	Opera	Chrome	IPhone	Android
OS 6.0	9	3.0+	3.0+	10.0+	3.0+	1.0+	1.0+

One of the ways we can check for canvas support dynamically can be done like.

```
try {
        document.createElement("canvas").getContext("2d");
        document.getElementById("support").innerHTML =
        "HTML5 Canvas is supported in your browser.";
        }
         catch (e)
        {
        document.getElementById("support").innerHTML =
        "HTML5 Canvas is not supported in your browser.";
 }
```

From the code above, we try to access the context after creation of our canvas. If the browser does not support the canvas element, the fallback text will be displayed. It should however be noted that if your browser supports the canvas element, it doesn't necessarily mean that it supports the text API thus that needs to be checked as well. However, if you don't want to write your own function for browser support, you can use open sources or libraries such as modernizer that helps check this. With this, the only thing you need to do is to include the script in the head section of your document. This will run automatically without the need to initialize. Accessing of pixels using the canvas element will therefore require you to be aware of the limitations involved which will mostly only allow you to access your data in the same domain as the script. If for some reason accessibility is limited, the need to run the application on a local or a remote web server will be necessary.

For my project, I deliberately did not use the excanvas, as I needed to see if my test users would be able to notice that canvas was not being supported in older versions of Internet Explorer. I however encountered the problem whereby my Chroma key part of the project wouldn't work outside my local host.

4.3 Styling webpages

One of the powerful technologies responsible of changing the look of how links, text and images as appear on a web page is known as Cascading Style Sheets (CSS). This technology combined with HTML allows rendering of beautiful pages. Like HTML, it has gone through a process of evolvement and has had different variations. It was around the mid 1990's when Cascading style sheet first was introduced to the World Wide Web Consortium [23]. At that time, Html pages were styled within tag attributes. Whereby, if you needed to make a heading text colour to blue, you would do something like this.

```
<h1 color = "blue">This heading is blue</h1>
```

At first, this was seen to be fairly simple. However, problems arose when the need to have the same look was needed for all the h1 tags. That would mean having to write individual styles for each of these tags as the styling was tied to the content itself. That would then leave us with a code that looked unpleasant to read. However, when Cascading Style Sheets gained popularity, global formatting of styles through the provision of a presentation layer that allows ease of modification, advanced options of design such as robust margin and padding possibilities as well as media targeting was made possible [18].

Regardless of the tremendous support from W3C, Internet Explorer still lacked full support for style sheets. However, with the introduction of various variations such as CSS1, CSS2, CSS2.1 and CSS3, versions of Internet Explorer 8+ have allowed full support of at least CSS2.1 syntax, meaning with time, they might be able to have full support of CSS3 which has added new capabilities and features as opposed to those defined in previous variations, over preserving backward compatibility [23]. All the new styles in CSS level 3 can now be applied in HTML5 elements although since it is also still under development, it has similar complications as HTML5 therefore, fallback use such as the modernizr: which is a JavaScript library that tests Html5 features can be used [16]. This allows one to add a combination of styles. An example of how this can be done is shown below.

```
/* Settings for browsers that support border-radius. */
.borderradius header {
border: thin #336699 solid;
border-radius: 25px;
}
/* Settings for browsers that don't support border-radius. */
.no-borderradius header {
border: 5px #336699 double;
}
```

Just having the code above it not enough. You will also need to have the no-js attribute to the root <html> element, which will generate a class of styles in the root [xx].

Types of style sheets

Including style sheets on a web page can be done in three different ways. These can be either through an external CSS file, locally on a web page or within the HTML element. Each of these ways can be used depending on how much content you need to display. When linking the style sheet from an external file, it is referenced within the head element as shown below.

```
<link href="style.css" rel="stylesheet" type="text/css"/>
```

With this kind of linking, you can reuse your style sheet as often as you want. A visual example of how it is done is shown in figure below.

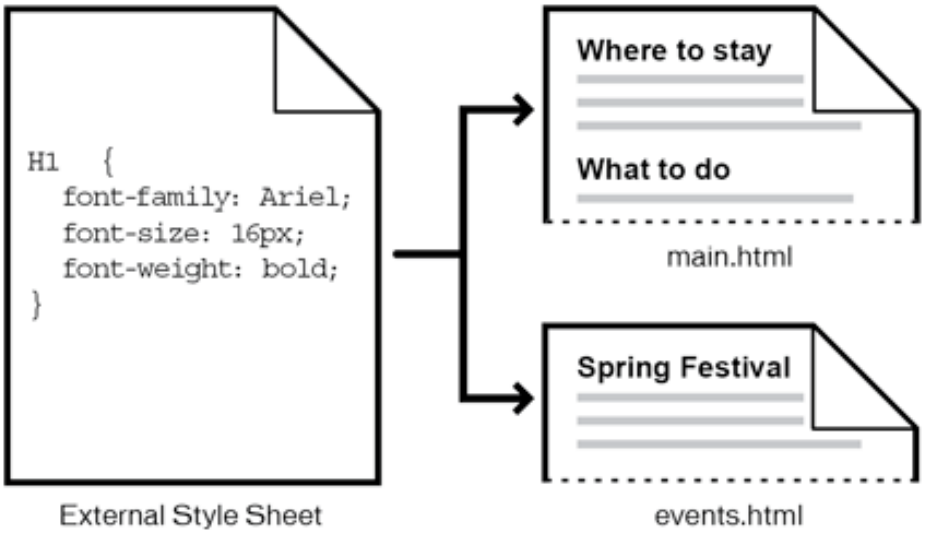

Figure 14. Including external styles on html pages

As seen from the figure above, we have a representation of how we are able to include all the styles in one single file and then reference them in our HTML file thus saving us time and eliminating the duplication process of the same code. This also becomes useful when thinking about having to cater for different screen sizes and resolution. With that, you need to create device specific styles for different device, though you can still

have them in one external file. A visual example of you can have multiple pages that share the same look and feel as indicated previously can be seen below.

Figure 15. Output of pages styled using CSS

Alternatively, when adding styles locally on a page, those styles are seen to be only specific to that page. Whereby if you need to have the same styles for a different page, you would have to write those styles to that page. Adding styles locally on the page is then seen to be necessary when you want to style a specific section that does not appear on any other page, changing the look of certain elements or when you have very little content to display that would make having an external file appear un-necessary. For example having a under construction page that would only need a line of text and maybe one image. An example of how you can specify styles locally on a page is shown below

```
<style type="text/css">
      #content { width: 350px; border: 2px solid #000; padding: 10px;}
              h1 { font: bold 12px Arial; color: blue;}
     p.justified { text-indent: 25px; text-align: justify; color: red; }
p.left-aligned { text-align: left; color: green; }
</style>
```

Creating styles for a page using the inline method is whereby you specify the styles within the HTML tag itself. As the inline style is specified within the tag itself, other tags on the page will not be affected. An example of this is shown below.

```
<p style=”padding: 0px;”>
```

In this example, if in case you had other styles either externally linked or embedded, the inline style gets the highest priority and the other two are overridden. Since cascading style sheets are just used for styling purposes in the project and are rather not necessary for the functioning of our player, I will not go deeper in it as it’s a wide subject area. I will however mention about some of its syntax and the box model that is somewhat necessary for the placement of our canvas elements.

Style sheet syntax

Cascading Style sheets are composed of selectors, properties and values. Each of the syntax is necessary in order to have a fully functional style sheet. The example below shows how its nomenclature is represented [24].

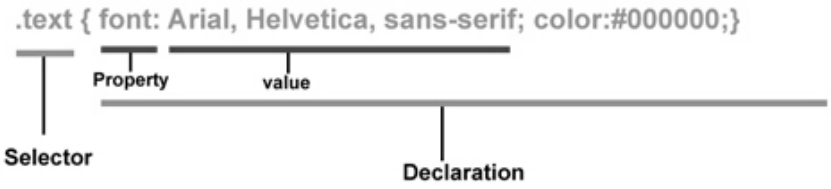

Figure 16. Anatomy of Cascading Style Sheets rule

Within the curly braces you can include as many declarations as you wish. And this can be represented as follows.

```
h1 {color: red; margin: 0; padding: 5px;}
```

Moreover, you can specify multiple selectors for any set of declarations in a comma-separated list:

```
h1, h2, h3, h4 { color: red; margin: 0; padding: 5px;}
```

When representing Cascading style sheets on a webpage, browsers follow a set of implementation guideline principles in CSS such as the cascading, inheritance and the specificity principle. With this principle, when having two identical rules, the one closest to the targeted element wins. In the case where we have

```
.p { color: red; }
.p { color: blue; }
```

The second rule, with the color declaration as blue would take the effect. Nevertheless, when we talk of the inheritance principle, the rules are based on nested tags like shown below

```
body {font-family: Verdana, Arial, Helvetica, sans-serif;}
```

The above rule uses the body tag as a selector and also sets the font family for every other text element on the page unless otherwise specified. As for the specificity principle, a class selector is seen as being more specific than that of a tag selector [23].

The box model

Cascading style sheets tend to treat everything as a box. This box represents the layout and placement of elements. Each of these boxes is separated either by margins, borders or padding's. Its implementation is one of the confusing things about CSS as

its placement can be really frustrating. With properties such as float, one would have to ensure the styles are consistent across browsers as this is one of the things can aid in element placement frustration [23]. This model tells how the elements and its attributes relate to each other. The arrangement of these boxes can be seen below.

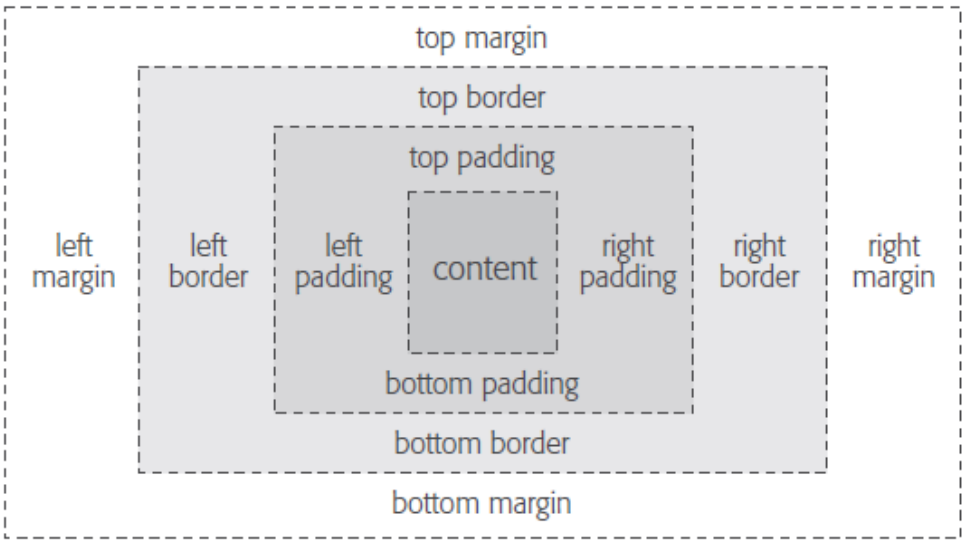

Figure 17. The box model

The properties above have assigned units in terms of pixels, ems, a percentage or a keyword inherit. The margin clears the area around the border while the padding clears the area around the content of the border. The width and height of elements greatly affects the spacing within the box model since you need to add the borders, padding and margins in order for you to calculate the size of a specific element. Firebug is one of the Internet add-on that allows a developer to see how the box is spaced and what size each element is [23].

Whenever the values of the margin, borders and padding are not declared, the value is always zero. As for the width, if none is declared and the element is either relatively or statically positioned, the value is always 100 percent. However, if the width is specified, then the element is pushed outwards thus making the actual size of the box to change. Floats and absolute positioning of elements on the other hand have no width declared and therefore behave in a different way. The content of the floated and absolute positioned element is therefore as big as the content unless a specific width is declared.

5 Design and Implementation

This chapter starts the second part of this thesis which focuses on the software implemented as the experimental part of the thesis project. The integral requirements of this thesis stated that the final implementation would provide a custom video player that would stream videos across browsers without the need of a third party plug-in. It would also have the ability to provide an online web based editor that will use JavaScript and canvas API to achieve the chroma key.

In order for the project to be easily extended, it was designed in a way that allowed developers to add their own functions and improve the video player functionalities. The project consisted of the following:

- HTML5 as our markup language,
- JavaScript API would be used to manipulate the player
- Canvas would be used to for graphics rendering and visual display.
- CSS would also be used for styling purposes.

The solution would allow users to play their videos in devices such as iphone and ipad that do not support flash due to recent decisions from apple not to support flash. Although the player was intended for use on the ipad and iphone, a flash fallback would be used to allow other users who were interested in viewing the videos with other devices that lacked browser support.

My main reason for choosing HTML5 video was because this would be an alternative for any operating system that did not support flash such as that of iphone and also because HTML5 allows far more customization on the fly than Flash ever could. It's accessible, it's standards compliant, and above all it's simple. The need to explore the new features also allowed me to test individual features across browsers.

Implementation of the project would demonstrate how HTML5 is capable of providing a means of displaying videos in specific formats across the web as well as the ability to manipulate the video using JavaScript and the canvas element.

5.1 JavaScript API

In order for us to implement a custom player, it is necessary to understand how the JavaScript API works since it is used extensively in the project. Main focus will be on the controls method in the API as that is what will be mostly required to create the player. This will help us in developing our player with custom controls. Further on, we will be able to manipulate the video using canvas to create an online-based video editor.

JavaScript is known under many names; the formal standard name is ECMAScript and proprietary variants include JavaScript, LiveScript and JScript [2]. It is known under many names; the formal standard name is ECMAScript and proprietary variants include JavaScript, LiveScript and JScript [2]. It is a vastly used programming language and probably the most popular script language. It is used in web browsers for client-side programming tasks to execute all kinds of tasks ranging from manipulation to the execution of an image analysis program. JavaScript overcomes the limitations of HTML and CSS by providing full flexibility to change anything in the Document Object Model (DOM) Programmatically. It has its own specified functions and should not be confused with the DOM API [12], [21].

JavaScript interfaces with HTML through the DOM (Document Object Model). The DOM is a hierarchical object structure that contains all the elements of a page with its attribute values and access functions. It represents the hierarchical structure of the HTML document and allows JavaScript to gain access to the HTML objects [21]. An example of how this works is shown below.

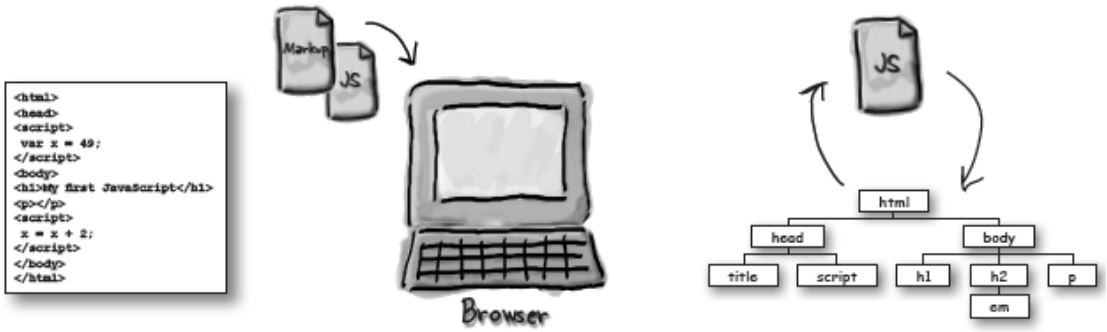

Figure 18. Document Object Model hierarchical structure representation

From the figure above, we can see that; an index.html file is first created then its loaded to the browser where it encounters the JavaScript and the browser parses the code and checks for any errors. If successful, JavaScript continues executing using the DOM to examine the page, change it, receive events from it or ask the browser to retrieve other data from the web server.

Features of media Elements

There are some General IDL attributes specific to the media resources that are handy when it comes to developing a custom control. These are: currentSrc, startTime, duration, volume, videowidth and videoHeight. Each of these features can be specified through content attributes either directly on the <audio> or <video> element or the selected source. More of these features will be seen in our application and how they are used.

The Media API is responsible for providing such features and it also explains how these features can be used. Control methods in the API help us know what functions are available as well as what kind of events can be triggered. Some of the JavaScript control methods defined in media elements include:

- play()
- canPlayType()
- load()
- pause()

These methods when executed on the media element perform a certain function. For instance, when the play() is executed, it sets @pause IDL attribute to false and starts playback of the media resource, downloading and buffering data as required. On the other hand, if the pause() is executed, the @pause IDL attribute is set to true and stops playback of the media resource. Each method undergoes a certain sequence of steps. The load() method when executed will cause all media resources to be suspended while allowing the loading process to be restarted. Nevertheless, the canPlay-

Type() will loop through the MIME types to see whether the browser can playback that MIME type.

The canPlayType() method takes a string as a parameter and is capable of returning three possible values, which are: -

- Empty string – Occurs when the browser cannot decode the media resource
- Maybe – This is when the browser can or cannot render the media resource.
- Probably – When the browser can decode and render the media resource in a media element because this implies knowledge about whether the codecs in a container format are supported by the browser.

Browsers are encouraged to only return "probably" for a MIME type that includes the codec's parameter [21]. I will not go into details of how each of the control methods work but I will instead discuss more of how you can use the API to create the custom player later in the chapter.

5.2 Implementing the video player

My application involved implementation of a video player that would have a consistent look and feel across all the browsers and consists of interface elements that aid in video control. The implementation would demonstrate the feasibility of the chosen approach and the fundamental idea underlying multimedia usage across browsers. This would then allow video to be manipulated using JavaScript and cascading style sheets to achieve a more native player to be treated as an alternative to flash or silverlight.

Tools and Software

Development of the project was mainly done on Macintosh for the fact that my day-to-day working environment was based on it. Other tools used for development included a text editor, video converters, which I did not use but could have been used included converters such as firefogg that converts .mp4, .mov or .flv format into .ogg and WedM; handbrake that allows conversions of .mov, .mp4 or h.264 and Riva FLV Converter would allow conversion to .flv format. Other converters such as Adobe encoder

would require payment of the license or allow trial usage. Different browsers would also be necessary for video support test purposes. Other requirements included installation of different browsers for testing purposes as well as a web server to host the project.

Programming Languages

Having studied a variety of languages, I was now able to extend my knowledge by using the latest technologies of which I had basic understanding. This would help me extend my basic Html knowledge to the new HTML5 that has new features as well as extend my JavaScript knowledge as I had done very little with JavaScript in my previous courses. The project development revolved around the HTML, HTML5 Video and canvas, JavaScript and styling using CSS and CSS3. The use of open source and permissive license libraries was also considered to avoid inventing the wheel twice as this would help in speeding up the development process as well as minimize bugs in the code.

Implementation

The first step is to have our folders ready and our index.html file. I created two folders. One folder was for the styles and another one for the scripts. The media resources were put in the same location as the index.html as I had problems loading the videos. I then downloaded the video files in three different formats: webM, .ogg and .mp4 format. Each of these formats was needed so that the browsers would be able to select a suitable format to play. A .flv format was used as a flashback however my .flv file was not the same video as the rest since I just needed it for test purposes. Once I had all my materials in place, I was able to begin the actual coding process.

First thing I did was to get my index.html file ready and included the necessary files that were required for the project development. In addition to the index file, I created an example file, which I used to create a basic video player using pure html5 elements and tested the video across different browsers to get an idea of how this video would render. A skeleton code was then created. The figure below shows the results of the basic video player that used the video tag and attributes to display it.

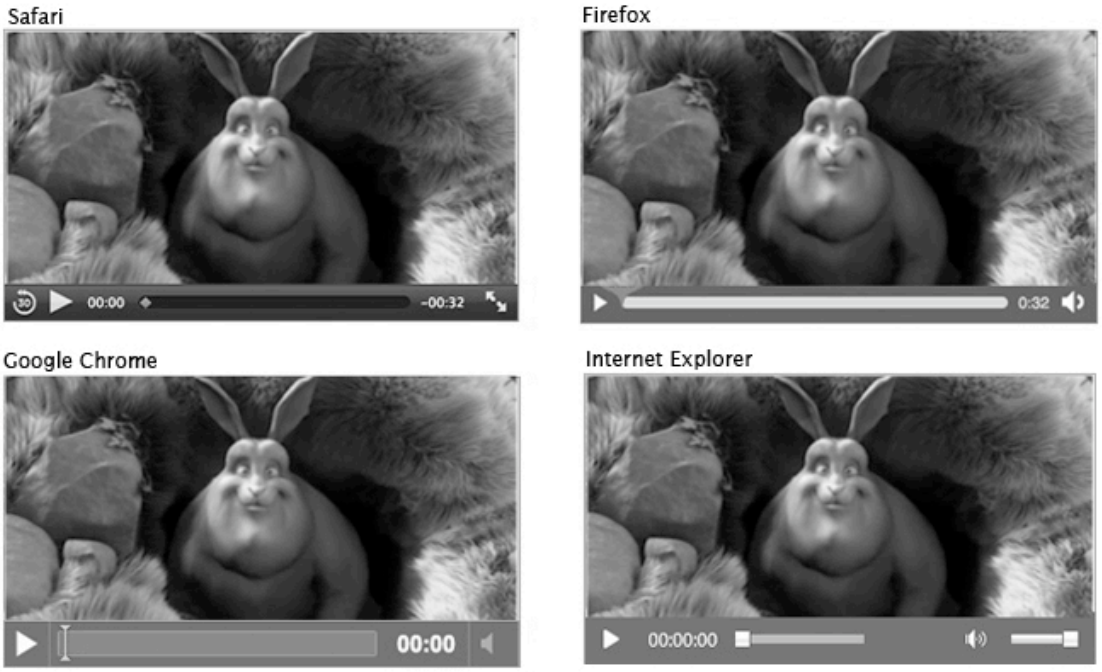

Figure 19. Video Support across browsers

As shown from the figure above, the basic player has different controls depending on the browser. Cascading style sheets are then seen to be handy when thinking of having a consistent look and feel. One this was done, I was now able to start with the main part of the project. I then opened my blank JavaScript file and defined the global variables and functions, which would be called when the page was fully loaded. An example of how these variables were created is shown below.

```
var html5VideoPlayer = {
            videoTypes  : ['mp4', 'ogg', 'webm'],
          defaultVideo  :['oceans-clip', ['mp4', 'ogg', 'webm']],
                mono    : false,
              volume    : 1.0,
                muted   : false,
```

After the variables are defined, we are then able to initialize the player. For this to happen, we will create a container to hold the video, the video controls and the canvas

element. We will then give a class for the container, as this will be useful when styling. The script below describes this.

```
init: function(params)
        {
        var videoContainer = document.createElement("DIV");
        videoContainer.className = 'videoContainer';
        this.videoContainer = videoContainer;
```

Once we have created those elements, we would then need to check what video formats are supported. And for this to happen, the script loops through each video formats to check whether a specific browser can play a certain type. The video object would then provide a method called the canPlayType, which will determine what video format will be played. As previously mentioned, this method returns one of the three different values: the empty string, the "maybe" or "probably". The figure below shows how this works.

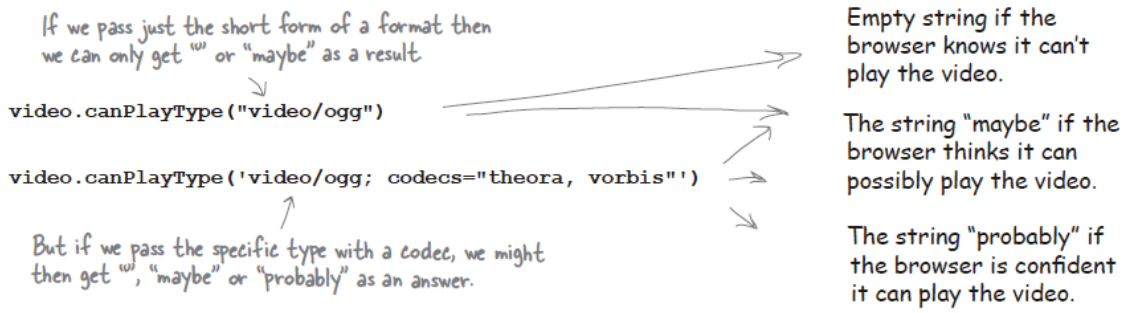

Figure 20. Representation of canPlayType control method

Sometimes the canPlayType event does not fire at once or at least when you expect it to. So another alternative for would be to listen to the loadstart event which is normally dispatched when the browser begins searching for media data as part of the resource. When the browser has paused fetching for media data and the entire resource has not been fully downloaded, browsers such as Firefox may fire the suspend event. This helps us to get the events fired from loadstart to loaded data then to the progress

after which it checks the canPlayType and when successful, it plays the video. Other events such as @error are dispatched when an error occurs while media is being fetched. An example of how the canPlayType method is called is shown below.

```
for(var i=0; i < this.videoTypes.length; i++){
    if(videoObject.canPlayType){
        if(videoObject.canPlayType("video/"+this.videoTypes[i]) == 'maybe' ||
          videoObject.canPlayType("video/"+this.videoTypes[i]) == 'perhaps')
         {
          this.videoTypesCanPlay.push(i) …
```

Once everything is successful, the video will need to have controls. For this, a video controls container was created to hold all the controls. Once this was in place, different controls such as play, mute, pause, volume and duration were created. Event handlers such as "ended" would be called whenever the video ended but not when it was paused. A reference to the video object which uses conditional statements would also be created whereby, if say the play button was pressed, the video would play; alternatively when the pause button was pressed, the video object pause method would be called hence the video would pause. As I progressed, I was also able to style the controls using CSS. The code below shows an example of how the play button control is created and appended to the control.

```
        var playButton = document.createElement("DIV");
                        playButton.appendChild(document.createTextNode("Play"));
                        playButton.addEventListener('click',this.play,false);
                        this.playButton = playButton;
                        this.buttonsPlay = playButton;
                        this.videoControlsContainer.appendChild(playButton);
```

After this, we then need to initialize the start playback function. This allows events to be fired whether the video is paused or not. From our example below, if the videoObject is say paused then we need to return the loading status back to the start of the video.

```
startPlayBack: function()
        {
          if(!this.videoObject.readyState) return false;
          if(html5VideoPlayer.videoObject.paused) return false
          if(this.videoObject.readyState <3 && !this.videoObject.paused)
          { this.bufferUU = true;
          }
           if(this.videoObject.readyState >= 3 && this.bufferUU)
          {
          this.bufferUU = false;
          this.videoObject.play();
          }
          if(this.videoObject.ended)
          {
  (html5VideoPlayer.videoLoop) ? html5VideoPlayer.videoObject.play()
      : clearInterval(html5VideoPlayer.startPlayBackInterval);
}
```

We have to keep polling the video until it is ready, otherwise we can not determine the duration, and hence can't create the seek control. When our video has completed its buffer or download, we are then able to see how much the video has loaded. This allows us to catch the ready State event. Once the video is fetched or before it's processed, our progress event allows us to know which event is triggered. When this event is dispatched, we can then update our search control. Our duration indicator will allow us to know the how much of our media resource has been downloaded. The picture below shows the result of our video player.

Figure 21. Custom video player

Once the video was working, the next step was to process the video in real time in order to add effects. This was done through manipulating the video's pixels and altering them to achieve a desired effect. The canvas element was then used, as it was the place where we would draw our output video. We'll start by creating a global variable named addEffects. This variable is going to hold a function that can take data from the video, and apply a filter to it. That is, depending on which effect we want, the addEffects variable will hold a function that knows how to process the video data and make it black and white, or mono, or inverted. An example code of our canvas object with its class, width and height defined is shown below

```
var canvasObject = document.createElement("CANVAS");
        canvasObject.className="manipulate";
        canvasObject.style.width = "720";
        videoObject.style.height = "400";
        this.canvasObject = canvasObject;
```

To begin the manipulation process, the use of the canvas draw pixel requires the direct access to the pixel buffer. With this, the canvas bounds pixel drawing to JavaScript. The first step is to retrieve the imageData object from the canvas as shown below

```
canvasImage = this.canvasObject.getContext("2d");
var imageData = canvasImage.getImageData(0, 0, this.videoWidth, this.videoHeight);
var pixelData   = imageData.data;
```

When it comes to the actual manipulation of our video pixels, what actually happens is that the video player decodes and plays the video behind the scenes. It is then copied frame-by-frame into hidden buffer canvas and then processed. We then go over the buffer pixel by pixel, passing each pixel to our effects function for processing by manipulating the RGB values in each pixel. This is shown below

```
for (var i = 0, n = pixelData.length; i < n; i += 4)
```

After the frame is processed it is copied into another display canvas to be viewed. An example of how this works is shown in the figure below.

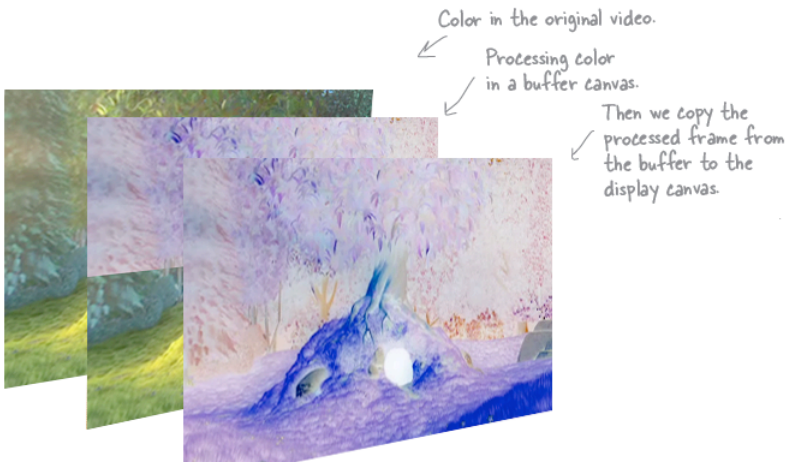

Figure 22. Processing color to canvas

From the figure above, the format of the pixel array returned by the call consists of four bytes of data, which as mentioned above are the RGBA channels. Each color is an integer between 0 and 255. These pixels are processed from left to right, top to bottom and start at index 0. In order to have the canvas element positioned properly, we have to position the video and the canvas on top of each other. The video element will be positioned at the bottom; the buffer on top of the video and the canvas element will be positioned on top of the buffer. To have this, we will use cascading style sheets. The code below shows how this is done.

```
.videoContainer .videoObject   { position : relative; …
.videoContainer video          { position: absolute; left: 0px; top: 0px;}
```

As you can see above, the video container is positioned relative to the element it in. and then from below the video canvas is positioned absolute with respect to the videoContainer. So by placing the canvas element 0px from the top and 0px from the left, they are exactly in the same position as the video and the video container.

```
. videoContainer .manipulate    { position: absolute; left: 0px; top: 0px;}
```

When it comes to the actual implementation of the video processing, we have to first consider the video element, the buffer canvas and the canvas element, which will display our final output. Each of these will be placed on top of each other as previously explained. To process the video, we will use the play event, which is called as soon as the video starts.

Implementation of the buffer would need us to take the video frame and copy it to the buffer canvas and then process the data in the frame. The drawImage() method will then take an image and draw it onto the canvas at position x,y for a given width and

height. It then gets the video element by specifying our clip as the source and then the drawImage() gets one frame as the imageData. This is shown below.

```
this.canvasObject.getContext("2d").drawImage
    (document.getElementsByTagName("VIDEO")[0],0,0,
    html5VideoPlayer.videoWidth,html5VideoPlayer.videoHeight);
 this.addEffects();
```

We then grab the imageData from the canvas context and store it in a variable frame so that we can process it. By defining the width and height, we show that we want the all the imageData in the canvas. An example of how this has been done is shown below. [11]

```
var imageData = canvasImage.getImageData(0, 0, this.videoWidth, this.videoHeight);
```

Once we have this in place, we are now ready to process the buffer. For this to happen we will loop through the every single pixel in the frame data and pull out the RGB values stored in each pixel and then we call the addEffects function. This code explains how this is achieved.

```
var imageData = canvasImage.getImageData(0, 0, this.videoWidth, this.videoHeight);
var pixelData = imageData.data;
```

First we find out the size of the frame.data, which is four times larger than the size of the canvas. Each pixel has red, blue, green and alpha values and this allows us to loop over the data and get the values for each pixel array. After that, we can grab red from the first position, green from second and blue from the third. An example of achieving

this whilst manipulating the color range to achieve a greyscale effect can be seen from the code below.

```
                for (var i = 0, n = pixelData.length; i < n; i += 4)
                {
                if(this.canvasGrayscale)
                {
    var grayscale = Math.floor((pixelData[i]+pixelData[i+1]+pixelData[i+2])/3);
 };
pixelData[i  ] = (this.canvasGrayscale) ? grayscale : pixelData[i];   // R
pixelData[i+1] = (this.canvasGrayscale) ? grayscale : pixelData[i+1];   // G
pixelData[i+2] = (this.canvasGrayscale) ? grayscale : pixelData[i+2];   // B
```

After this, we call the addEffects function and pass the position of the pixel, the RGB values and the frame.data array. The addEffects function then updates the frame.data array with new pixel values and have them processed according to functioned assigned to the addEffects. Once this is done, we have the frame.data processed so we use the context putImageData method to put the data into the display canvas. This method takes the data in the frame and writes it onto the canvas at the specified position [11]. This is shown below.

```
canvasImage.putImageData(imageData, 0, 0);
```

Our output will then look something like this.

Figure 23. Greyscale effect

5.3 Video editing using the Chroma key

Online video editing is a relatively new technology. Currently, video editors use software such as Adobe flash and HTML5. However, since HTML5 is still new, not so many people are using it. I decided to create an online-based video editor focusing on the chroma key by using JavaScript and canvas to manipulate my video. My main aim for creating this was because there are few video editors available online and also because we lack web-based video editors that allow users of novice skill level to demonstrate their ideas visually through a multimedia composition.

Another term for the chroma key is blue screen. It is based on luminance key and works in a way that selects all the green pixels in the video and replaces them with the desired image. While previously adding effects to our video, we can use the same principle but this time we only pick the green pixels so when we run the effect and click on the image, the video background changes its background from green and adapts the image as a background. The chroma key creates color on just one key channel [26].

For this part of the project, I decided to use a separate video. I was able to download a video with a green background from YouTube so that I would be able to pick the green pixels and get to see a clear difference. A HTML5 video editor was then created to test the hypothesis that using an editor without a plug-in is possible.

To begin creating this, I had to research on the different types of online-based editors to try and understand how the process works. I also had to look into what codecs would be suitable for carrying out this part of the project. For my project to be a success I had to get a video with a green background. This would allow me to replace all the green pixels with another image easily. Once I had my video, I needed to create two canvases. One canvas was to be used for the output while the other was used to display the current frame of the original video.

The Diagram below shows the results of the Chroma key. The image below shows the video, the actual image and the canvas image on the right. With our Chroma key, you are able to select any color from our video clip and remove it from the image and this creates a hole from where the color used to be which will then be replaced with our canvas image.

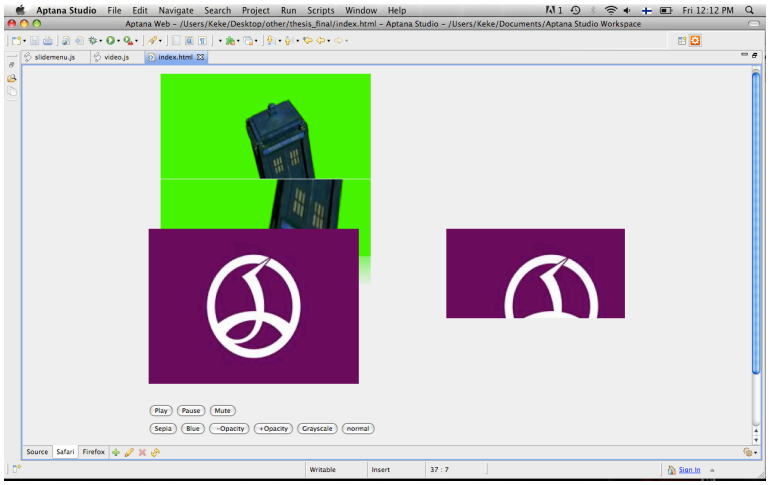

Figure 24. Chroma key before editing

The next image shows the edited video using the Chroma key effect. Our new image is played behind the image thus making it visible through this hole. An example of this is seen on the television whereby the weatherman is giving his nightly news. He or she is usually standing in front of a green wall and point at different parts of the country on a virtual map behind them. The video is totally replaced with the canvas image that was created and not the actual image. For this to take place. I had to click on the created canvas in order to achieve the results below.

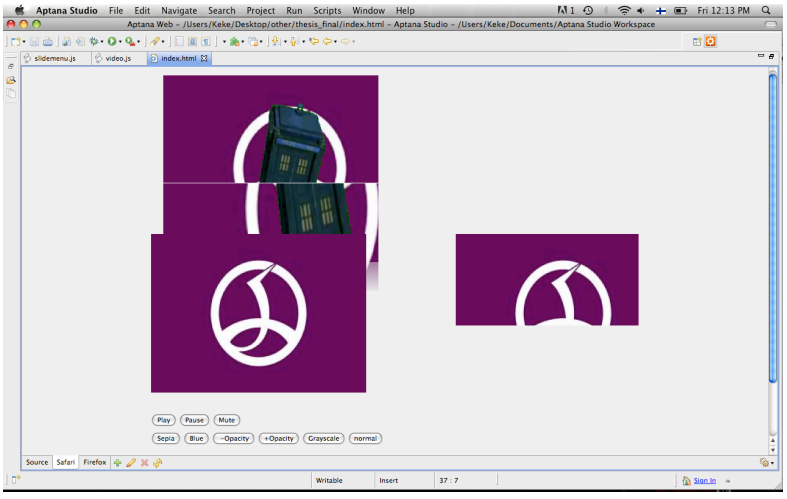

Figure 25. Chroma Key output

My chroma key example would only play on the local host due to some JavaScript security concerns. Once everything was ready, the video was styled using styles.

6 Testing and Results

The test hypothesis is that our HTML5 video player and editor will be a more satisfying product compared to other players that depend on third party plug-ins to effectively communicate ideas visually.

The main reasons for carrying out this test included establishing and identifying potential design concerns as well as user performance. This would allow effective collection of data and allow me to address any issue that arose. Its objectives were to determine design conflicts and usability problem areas with potential sources of errors such as browser support failure as well as provision of user satisfaction.

The main areas that were taken into consideration include:

- Installation of third party plug-ins if needed
- Playing video on different browsers to check browser compatibility
- Differentiation between html video player and other video players
- Support on various devices, in particular, ipad

The method used was expert evaluation according to Nielsen guidelines for conducting heuristic evaluation and the participants were able to use the following browsers:

- Mozilla Firefox
- Internet Explorer
- Google Chrome
- Opera
- Apple Safari

In this particular test, there were two participants who were required to fill out a comparison form based on the two provided players. The tables 7, 8 and 9 below shows the testing environments used by the participants. Both of the participants tested on all three mentioned platforms to try out the demo.

Table 7 below displays the specifications of Dell platform that was used in this test

Table 7. Specifications for Intel Dell

Computer Platforms	(Intel) Dell Pentium 4 CPU, 1.00GB RAM
Screen resolution:	1024 X 768
Operating system:	Windows XP,
Connection speed:	100.0Mbps

Table 8 below displays the specifications of Macintosh platform that was used in this test

Table 8. Specifications for Mac book pro

Computer Platforms	(Intel core) Mac book Pro, 2 GB RAM
Screen resolution:	1024 X 800
Operating system:	Mac OSX Tiger
Connection speed:	100.0Mbps

Table 9 below displays the specifications of ipad

Table 9. Specifications for Ipad 2

Platform	Apple Ipad 2, WI-FI + 3G
Screen resolution:	9.7 inch LED display, 1024 x 768
Operating system:	Mac OSX v.10.5.8
Connection speed:	100.0Mbps

Two tests were conducted during the course of this experiment. The orders of the tests were assigned to each person randomly. The two tests contained identical material and required them to perform certain tasks.

6.1 Methodology

The method used was expert evaluation according to Nielsen guidelines for conducting heuristic evaluation. Each participant was first required to fill out the form that was given to the asking some general questions before proceeding with the usability test. The test then began by separating the participants individually. Participant A, started by testing a basic player then went on how the actual demo player functions in comparison to plug-in based players such as YouTube across browsers then proceeded on testing the demo on different platforms. He had to change monitors for this to work. Since everything was already setup, this saved time and he was therefore able to try the demo on all the three platforms. While he performed these tasks, I was able to observe his facial reactions and write down some notes on what I observed.

Once he was done with the first part of the task, he moved to the second one, which was to try out real-time editing online. For this task, I had free trial of adobe premier, as I could not afford purchasing one for my demo application. I had a sample video for which he would use to edit. The task instructed him to try video manipulation. He was either to add some text on the video or change color. He tried both and once he was done with the testing. I then interviewed him and asked oral questions as to what he thought about the tasks in general. When I was done with the questions, I allowed him to give suggestions to what he thought he would have loved to add or remove in my demo application. After we were done, the same process was done to participant B and results were recorded.

Later in the day, I was able to bring together both participants and asked them about their overall experience and then showed them how the tasks were to be done. I was also able to tell them what I had concluded and the results that I was able to collect from this test. It should however be noted that the test results are not fully reliable as there were some technical problems with the demo application at the time of the test. All in all, evaluation took about half an hour for each participant and was carried out two times with one person per test. Both participants did not have knowledge of the new features of html5 video even though one was familiar with the Internet.

6.2 Results

The results in this report were formulated based on notes taken and observation while executing the test and interviews taken after the test. The table below shows a summary of the problems encountered by the participants.

Table 10. Consistency problem

Consistency	Suggestions
It was found that a basic html5 player lacked consistency of the look and feel as the controls differed from browser to browser.	It was suggested that the use of Cascading style sheets would help in creating a consistent look similar to the one provided to the actual demo application.
It was also found that the styled custom player was consistent across all browsers.	It was suggested that styles should always be used as they help improve the look and feel of the components.

Table 11. Match between system and real world conventions

Problem	Suggestions
The controls bar should disappear when the mouse is placed over it.	It was suggested that the demo application should follow real-world conventions, making information appear in a natural and logical order.
Lack of upload and download function or drag and drop function	It was suggested that it would be a good idea to have an upload function in order to be able to edit other videos of users' choice and download once they finished editing them

Table 12. Visibility of Status

Browser Support	Suggestions
It was found that html5 video is not supported in older browser versions. And flash is not supported on ipad	It was suggested that a download link to be provided for users who wanted to view the video but lacked flash It and that publishers should at least publish videos in two formats. Flash and HTML video. It was also suggested that a message indicating lack of support would be helpful to users who lacked support.
It was also found that some of the features like the progress bar was visible on other browsers but not on others.	It was suggested that maybe the use of a script or an open source library would be useful to fix this as that would allow consistency as well as help the users see how the video playback is progressing.

The first form that was given to the participants was used to establish their level of competency when using computers. The information collected was then used to determine whether the participant's knowledge had any effect with the amount of time it took for them to complete the tasks. From the data collected, it was evident that the more interaction a participant had with computers on a day-to-day basis the faster it was for him or her to completion the given tasks. This becomes relevant when looking at how users of little or no experience can pick up the software and begin working and the simplicity of our video editor compared to the other editors, which require training.

When it came to the editing part of the video, the data extracted from this determined the problems that arose whilst manipulating content. Later on, the participants were asked to fill out another form and give their feedback. All these data was collected in a more direct approach to finding what editor that the participants preferred. The participants were then asked to indicate their preferred choice of editor, between the demo application and the adobe premier in terms of functionality and overall ease of use.

It should be noted that changing the order in which the participants took the tests was less biased and prevalent. In addition, the random order of which test the participants received and the results from the experiment provided me with neutral information. The table below shows a summary of results.

Table 13. Summary of results.

Task	Result
Overall performance	It was easy for the participants to manipulate the demo video. The Adobe premier was not easy to use and this took more time for the participants.
Functionality	The custom player was seen to be easy and only consisted of effects. It was suggested that it would be a good idea to develop the player with more features.

7 Conclusion

During the course of the research of this thesis, it was discovered that HTML5 is also much more stable than Flash and may be a rather large security risk (Jobs, 2010). Flash is the number one reason for Macs crashing and Symantec gave Adobe Flash one of the worst security records in 2009 (Jobs, 2010). This may mean that using HTML5 not only provides a more stable platform for web technologies to run off of, but also lessens the risk that users face when surfing the web. The incompatible codec support currently makes it harder than using plug-ins to simply embed video in a page and have it work cross-browser. Regardless of this, I was able to achieve the goal that was set.

At first, this project was a quite demanding for me, as I was not proficient in JavaScript and for this, I had to do a lot of reading. I, however, managed to create a fully function-able player that is supported in all the latest browsers. I can now say that I have learnt how to program using JavaScript though I would not say that I am a professional but I am sure I will get there someday. I have also come to learn that one can create amazing applications using JavaScript, canvas and cascading stylesheets. In future, I plan to continue developing the project further and have newer functionalities.

This thesis has also allowed me to learn how to use open sources and how they can aid in workload reduction. Additionally, I have been able to appreciate them as they help one get an understanding of what happens within a specific script and give you a lead on where to start if you want to develop a certain application. I can now say that my application can be used as foundation basis for those who intend to develop the application further. I have been able to achieve the goal set for this thesis, which was to provide a functional custom player that would work across the current web browsers and allow support of newer devices that have lacked support for third party plug-in such as flash.

The project was interesting and I have enjoyed working on it. I have spent a lot of time working on it but I believe it is best to try get to get a thorough understanding of what one is actually doing.

References

1. CERN. *The Website of the world's first-ever web server*, 2008. Available at: <http://info.cern.ch/>[Accessed: 13 May 2011].

2. Crockford, D. *Douglas Crockford's Wrrrld Wide Web RFC 2396*, 2001. Available at: < http://www.crockford.com/javascript/javascript.html > [Accessed: 14 September 2011].

3. David, M. 2010. *HTML5: designing rich Internet applications*. Amsterdam, Focal Press.

4. Dave Ragget, *Ragget*, Jenny Lam, Ian Alexander and Michael. *4*, Addison Wesley Longman 1998. Available at: <http://www.w3.org/People/4/book4/ch02.html> [Accessed: 16 February 2011].

5. Dave Ragget, Arnauld Le Hors, and Ian Jacobs, eds. *HTML+ (Hypertext markup format), working draft, HTML+ Discussion Document* 8 November 1993 Available at:<http://www.w3.org/MarkUp/5/5_1.html> [Accessed: 16 February 2011].

6. Dave Ragget, *HyperText Markup Language Specification Version 3.0*, Internet Draft <draft-ietf-html-specv3-00.txt>,expired 28 September 1995. Available at: <http://www.w3.org/MarkUp/html3/CoverPage>. [Accessed: 16 February 2011].

7. Dave Ragget. *HTML 3.2 Reference Specification*, W3C Recommendation, 1997. Available at:<http://www.w3.org/TR/REC-7>.[Accessed: 16 February 2011].

8. Dave Ragget, Arnauld Le Hors, and Ian Jacobs, eds. *HTML 4.0 Specification*, W3C Recommendation, 24 April 1998. Available at: <http://www.w3.org/TR/1998/REC-html40-19980424>. [Accessed: 16 February 2011].

9. Dave Ragget, Arnauld Le Hors, and Ian Jacobs, eds. *HTML 4.0 Specification*, W3C Recommendation, 18 December 1997. Available at: < http://www.w3.org/TR/REC-html40-971218/>. [Accessed: 16 February 2011].

10. Dave Ragget, Arnauld Le Hors, and Ian Jacobs, eds. *HTML 4.01 Specification*, W3C Recommendation, 24 December 1999. Available at: < http://www.w3.org/TR/1999/REC-html401-19991224/>. [Accessed: 16 February 2011].

11. Freeman, E and Robson, E. (2011). *Head First HTML5 Programming*. CA, O'Reilly.

12. Guisset, F. *The DOM and JavaScript,* Mozilla Developer 2005. Available at: <https://developer.mozilla.org/en/The_DOM_and_JavaScript> [Accessed: 23 September 2011].

13. Keith, J. (2005). *DOM scripting: web design with JavaScript and the Document Object Model*. Berkeley, CA, Friendsof.

14. Lawson, B., & Sharp, R. (2011). *Introducing HTML5.* Berkeley, CA, New Riders.

15. Lennon, J. (2010). Create Modern Web sites using HTML5 and CSS3. [online] IBM cooperation. < http://www.ibm.com/developerworks/web/tutorials/wa-html5/wa-html5-pdf.pdf> [Accessed: 16 February 2011].

16. MacDonald, M. (2011). *HTML5 The missing manual*, CA, O'Reilly.

17. Lubbers P, Albers B, Salim F. (2010). *Pro HTML5 Programming*. Berkeley, United States of America: Springer Science+Business Media: Paul Manning.

18. Lowery J, Fletcher M. (2010). *HTML5 24 Hour Training*. United States of America: Wiley publishing Inc

19. Paul, K. (June 16, 2009). *HTML 5: Could it kill Flash or Silverlight? | Developer World. InfoWorld*. Available at: <http://www.infoworld.com/d/application-development/HTML5-could-it-kill-flash-and-silverlight-291> [Accessed:6 October 2011].

20. Paul, R. (2009). *HTML 5 and Web video: freeing rich media from plugin prison*. Available at:<http://arstechnica.com/open-source/news/2009/05/google-dailymotion-endorse-html-5-and-standards-based-video.ars> [Accessed:12 May 2011].

21. Pfeiffer S. (2010). *The Definitive Guide to HTML5 Video*, United States of America: Springer Science+Business Media: Paul Manning.

22. Pilgrim, M. (2010). *HTML5: up and running*. Sebastopol, CA, O'Reilly.

23. Powers, D. (2009). *Getting Started with CSS*, CA, Friendsof.

24. Sanders, B. (2011). *Smashing HTML5*, John Wiley and Sons.

25. T. Berners-Lee, D. Connolly, *Hypertext Markup Language - 2.0*, RFC 1866, November 1995. Available at: <http://www.ietf.org/rfc/25.txt> [Accessed: 16 February 2011].

26. Bradford, S, *The Blue/ Green Screen*, 1995 - 1998. Available at: <http://www.seanet.com/~bradford/blue_green_screen_visual_effects_1.html>

27. Tim Bray, Jean Paoli and C. M. Sperberg-McQueen, editors. *The Annotated XML 1.0 Specification*. 1998, O'Reilly Media, Inc., 1998. Available at: < http://www.xml.com/axml/testaxml.htm>[Accessed: 10 February 2011].

28. WHATWG. *WHATWG Wiki - FAQ.*, 2010. Available at: <http://wiki.whatwg.org/wiki/FAQ> [Accessed: 16 May 2011].

29. Dave Ragget. *HTML5 Tables - FAQ.*, 1996.

Available at: < http://ietf.org/rfc/rfc1942> [Accessed: 16 February 2011].

30. François Yergeau, Gavin Thomas Nicol, Glenn Adams and Martin J. Dü rst. IETF. *Internationalization of the Hypertext Markup Language*, 1997. Available at: <http://ietf.org/rfc/rfc1942> [Accessed: 16 February 2011].

31. Ben Adida, Mark Birbeck, *Steven Pemberton. Support for RDF in HTML4 and HTML5*, 2011. Available at: < http://dev.w3.org/html5/rdfa/> [Accessed: 28 April 2011].

32. Erik Dahlström, et al., *Scalable Vector Graphics (SVG) 1.1 (Second Edition*), 2011. Available at: < http://dev.w3.org/SVG/profiles/1.1F2/publish/> [Accessed: 28 April 2011].

33. Neuberg, B, 2009. *Introduction to HTML5*. [video online]Available at: <http://www.youtube.com/watch?v=siOHh0uzcuY> [Accessed: 2 April 2011].

34. Hickson, I, & Hyatt, D, 2010. HTML5, A vocabulary and associated APIs for HTML and XHTML, W3C Working Draft 4 March 2010. Available at: <http://www.w3.org/TR/html5/> [Accessed: 10 March 2011].

35. Hammond, J, 2010. *Does HTML5 Herald the end of RIA Plug-Ins? Not Really*. Available at: <https://www.adobe.com/content/dam/Adobe/en/industryinsights/solutions/pdfs/html_5_ria_plug_ins.pdf > [Accessed: 10 March 2011].

36. Software Product Labs Blog, 2011. *The benefits of HTML5 are growing more apparent*. [online]Available at: < http://blog.ness.com/spl/bid/70821/The-Benefits-of-HTML5-Are-Growing-More-Apparent> [Accessed: 2 December 2011].

37. Schmelzer,R, 2009. *The Dissolution of the Rich Internet Application (RIA) Market*. [online]Available at: < http://www.zapthink.com/2009/09/03/the-dissolution-of-the-rich-internet-application-ria-market/> [Accessed: 27 March 2011].

38. Ottoson,S, 2011. *Analyse of Interactive Graphics in HTML5 from view of practical application* Available at: <http://www.nada.kth.se/utbildning/grukth/exjobb/rapportlistor/2011/rapporter11/huovinen_ottosson_stefan_11029.pdf/> [Accessed: 27 November 2011].

39. Granbäck,C, 2009. *Rich Internet Applicarions (RIAs)*.Available at: <http://publications.lib.chalmers.se/records/fulltext/116839.pdf> [Accessed: 27 November 2011].

Appendix

Section A – General Test Questions

Please fill out the pre-survey questions below:

1. Please indicate your gender:
    - M
    - F
2. On average, what is your daily hourly use of computer or computer related tasks?
    - 1hr or less
    - 1-1.9hrs
    - 2-3.9hrs
    - 4-5.9hrs
    - 6hrs+
3. Do you consider emerging web technologies as a personal interest or hobby?
    - Y
    - N
4. Please list any video editing software you may have used in the past or currently use (Premiere, Final Cut, etc., or None)

5. Please list any computer graphic related programs you may have used in the past or currently use (Photoshop, 3D max, AutoCAD, etc., or None):

6. What is your primary browser
    - Internet Explorer
    - Apple Safari
    - Opera
    - Google Chrome
    - Mozilla Firefox

7. What is your computer operating system
    - Mac OSX
    - Windows
    - Linux

Section B - Test Tasks

1. You will find that two versions of the player are opened in separate tabs.
   - Compare and State their differences and similarities.
   - Play the video and test both players in Mozilla Firefox, Google chrome, Internet Explorer 6+, Opera and Safari. State your findings.

2. Close the video in the first tab. You now have the second video still opened. Open Adobe premier. Change the effects of our custom player and play with it to achieve different effects. Click where appropriate. When done, use Adobe Premier editor to reproduce similar effects you got from the custom player. The necessary files are already provided for you. You may refer to the "Custom player" as many times as you would like while editing.

3. Fill out the survey below in reference to the Adobe premier Editor: On a scale of 1 [strong disagree] - 3 [strongly agree])

   - Adobe premier was intuitive and easy to learn. 1 2 3 4
   - Had a layout that made it difficult to complete some tasks. 1 2 3 4

4. Please indicate which editor was better in the given situation by crossing the editor.

   - Overall functionality [ ] Adobe premier [ ] Custom Edior
   - Overall ease of use [ ] Adobe premier [ ] Custom Edior

You have completed the testing of our custom player. Thank you!

Section C – Questionnaire after exercise

1. Did you find using the tools as learning aids easy or difficult?

- Very easy
- Moderately easy
- Moderately difficult
- Very difficult

2. Please describe any difficulties that you experienced

.....

.....

3. Did you have any problems in the tools on certain platforms or in certain locations?

- No problems
- Minor problems
- Major problems.

4 What were the problems that you experienced, if any?

.....

.....

5. Do you have any suggestions as to how the tools could be made more accessible?

.....

.....

6. Do you have any other comments, criticisms or suggestions relating to the usability - ease of use - of the tools?

.....

.....

Source Code

```
		var html5VideoPlayer =
		{
		videoTypes		: ['mp4', 'ogg', 'webm'],
		videoTypesCanPlay	: [],
		defaultVideo		: ['bbb_trailer', ['mp4', 'ogg', 'webm']],
		videoWidth		: false,
		videoHeight		: false,
		videoDuration		: false,
		videoLoop		: true,
		bufferUU		: false,
		canvasOpacity		: 255,
		canvasInvert		: false,
		canvasGrayscale		: false,
		mono			: false,
		volume			: 1.0,
		muted			: false,
		controlsFocus		: false,
		controlsHide : true,

		init: function(params)
		{
/* Create container for video and controls */
		var videoContainer = document.createElement("DIV");
		videoContainer.className = 'videoContainer';
		this.videoContainer = videoContainer;

 /* Create video element */
		var videoObject = document.createElement("video");
		this.videoObject = videoObject;
/* Create canvas element */
```

```
                var canvasObject = document.createElement("CANVAS");
                canvasObject.className="manipulate";
                this.canvasObject = canvasObject;

                 /* Check supported formats */
                for(var i=0; i < this.videoTypes.length; i++)
                {
                if(videoObject.canPlayType)
                {
                if(videoObject.canPlayType("video/"+this.videoTypes[i]) == 'maybe' ||
                 videoObject.canPlayType("video/"+this.videoTypes[i])    ==    'perhaps')
                {
                this.videoTypesCanPlay.push(i);
                                }                                                    }
                                                }

/* Browser doesn't support any formats */
                if(this.videoTypesCanPlay.length <= 0) return false;
                videoObject.className = 'videoObject';
                videoOject.setAttribute("src",this.defaultVideo[0]+"."
                +this.defaultVideo[1][this.videoTypesCanPlay[0]]);

                videoObject.setAttribute("autobuffer", "true");
                videoContainer.appendChild(videoObject);
                videoContainer.appendChild(canvasObject);
                document.body.appendChild(videoContainer);

 /* Set up video width and height */
  this.initControlsInterval = setInterval('html5VideoPlayer.initControls()', 100);

                },
```

```
		initControls: function()
		{
		if(!this.videoObject.readyState) return false;
		clearInterval(this.initControlsInterval);
		this.videoWidth = this.videoObject.offsetWidth;
		this.videoHeight = this.videoObject.offsetHeight;

/* Set up canvas width and height */
		this.canvasObject.setAttribute("width",this.videoObject.offsetWidth);
		this.canvasObject.setAttribute("height",this.videoObject.offsetHeight);

 /* Create video controls */
		var videoControls = document.createElement("DIV");
		this.videoControlsContainer = videoControls;
		videoControls.onmouseover = function()
		{
		html5VideoPlayer.controlsFocus = true;};
		videoControls.onmouseout = function(){
		html5VideoPlayer.controlsFocus = false;};
		videoControls.className = 'videoControls';
		videoControls.style.width = this.videoObject.offsetWidth+ "px";
		this.videoContainer.appendChild(videoControls);

		 /* Play/Pause button */
		var playButton = document.createElement("DIV");
		playButton.appendChild(document.createTextNode("Play"));
		playButton.addEventListener('click',this.play,false);
		this.playButton = playButton;
		this.buttonsPlay = playButton;
		this.videoControlsContainer.appendChild(playButton);
		 /* Played/Duration */
```

```
                var currentTime = document.createElement("DIV");
                this.currentTime = currentTime;
                this.videoControlsContainer.appendChild(currentTime);

                /* Set up animations for the controls (hide/show) */
                this.controlsHideTimeOut=
                setTimeout("html5VideoPlayer.hideControls()", 5000);
                this.videoContainer.onmousedown= function()
                {
                html5VideoPlayer.hideControls();                                clear-
Timeout(html5VideoPlayer.controlsHideTimeOut);                                  }
                  /* Duration indicator */
                var durationIndicator = document.createElement("CANVAS");
                durationIndicator.className = "indicator";
                durationIndicator.setAttribute("WIDTH",this.videoObject.offsetWidth);
                durationIndicator.setAttribute("HEIGHT", 10);
                this.durationIndicator = durationIndicator;
                durationIndicator.style.background = "rgba(0,0,0,0.5)";
                this.videoControlsContainer.appendChild(durationIndicator);
                this.durationIndicator = this.durationIndicator.getContext("2d");
                durationIndicator.onclick = function(e)
                {
                if(!e) var e=window.event;
                var x = e.pageX;
                html5VideoPlayer.videoObject.currentTime
                        =(Math.floor(html5VideoPlayer.videoObject.duration)
                         /html5VideoPlayer.videoWidth)*x;
                }
                this.startPlayBackInterval= setInterval

                ('html5VideoPlayer.startPlayBack()', 30);
```

```
            },
            startPlayBack: function()
            {
            if(!this.videoObject.readyState) return false;
            if(html5VideoPlayer.videoObject.paused)return false;
            if(this.videoObject.readyState<3&& !this.videoObject.paused)
            {
            this.bufferUU = true;
            }
            if(this.videoObject.readyState >= 3 && this.bufferUU)
            {
            this.bufferUU = false;
            this.videoObject.play();
            }
            if(this.videoObject.ended)
            {
            (html5VideoPlayer.videoLoop)?
            html5VideoPlayer.videoObject.play()
            :clearInterval(html5VideoPlayer.startPlayBackInterval);
            }
            this.canvasObject.getContext("2d").drawImage
            (document.getElementsByTagName("VIDEO")[0],0,0,
            html5VideoPlayer.videoWidth,html5VideoPlayer.videoHeight);
            this.addEffects();
            html5VideoPlayer.currentTime.innerHTML=
            this.videoObject.currentTime.toFixed(2) +
            " / " + this.videoObject.duration.toFixed(2) + " |
            buffer underrun : " + this.bufferUU; //this.videoObject.paused;

// Update progress bar
            this.durationIndicator.clearRect(0,0,this.videoWidth,this.videoHeight);
```

```
            this.durationIndicator.fillStyle = "rgba(255,255,255,0.2)";
            this.durationIndicator.beginPath();  // Update progress bar buffer
            var buf = this.videoObject.buffered.end(0);

            this.durationIndicator.fillRect(0,0, (this.videoWidth/
            this.videoObject.durati   on.toFixed(2))*buf.toFixed(2),10);
            this.durationIndicator.closePath();  // Update progress bar progress

            this.durationIndicator.moveTo(0,0);
            this.durationIndicator.fillStyle = "rgba(200,30,220,0.75)";
            this.durationIndicator.beginPath();

            var buf = this.videoObject.buffered.end(0);
            this.durationIndicator.fillRect(0,0,(this.videoWidth/
            this.videoObject.duration.toFixed(2))*
            this.videoObject.currentTime.toFixed(2),10);
            this.durationIndicator.closePath();
            },
            play: function()
            {
               if(html5VideoPlayer.videoObject.paused)
            {
            html5VideoPlayer.videoObject.play();
            html5VideoPlayer.playButton.innerHTML = 'Pause';
            }                              else{
            html5VideoPlayer.videoObject.pause();
            html5VideoPlayer.playButton.innerHTML = 'Play';
            }
},
            addEffects: function()
            {
```

```
canvasImage = this.canvasObject.getContext("2d");
var imageData = canvasImage.getImageData(0, 0, this.videoWidth,
this.videoHeight);
var pixelData = imageData.data;
for (var i = 0, n = pixelData.length; i < n; i += 4)
{
if(this.canvasGrayscale)
{
var grayscale =
Math.floor((pixelData[i]+pixelData[i+1]+pixelData[i+2])/3);
};
pixelData[i  ] = (this.canvasGrayscale) ? grayscale : pixelData[i];
pixelData[i+1] = (this.canvasGrayscale) ? grayscale : pixelData[i+1];
pixelData[i+2] = (this.canvasGrayscale) ? grayscale : pixelData[i+2];

if (this.mono)
{
var monoColor =
Math.floor((pixelData[i]+pixelData[i+1]+pixelData[i+2])/3);
monoColor= (monoColor > 128) ? 0 : 255;
pixelData[i] = pixelData[i+1] = pixelData[i+2] = monoColor;
};
if(this.canvasInvert)
{
pixelData[i  ] = 255 - pixelData[i  ];
pixelData[i+1] = 255 - pixelData[i+1];
pixelData[i+2] = 255 - pixelData[i+2];
};


pixelData[i+3] = this.canvasOpacity;
```

```
pixelData[i+3] = 255-pixelData[i+2];
}
canvasImage.putImageData(imageData, 0, 0);
                },
blackAndWhite: function()
{
this.mono = (!this.mono) ? true : false;
                },
invert: function()
{
this.canvasInvert = (!this.canvasInvert) ? true : false;
this.addEffects();
},
toggleGrayscale: function()
                {
this.canvasGrayscale = (!this.canvasGrayscale) ? true : false;
this.addEffects();
},
changeOpacity: function(amount)
                {
this.canvasOpacity+=(this.canvasOpacity+amount >= 0 &&
this.canvasOpacity+amount <=255) ? amount : 0;
document.getElementById("canvasOpacity").innerHTML =
this.canvasOpacity;
this.addEffects();
},
addjustVolume: function(amount)

                {
var amt = this.volume; //.toFixed(1);
amt+=(this.volume+amount >= 0 && this.volume+amount <=1) ?
```

```
                amount : 0;
                this.volume = parseFloat(amt.toFixed(1));
                this.videoObject.volume = this.volume;
                document.getElementById("videoVolume").innerHTML =
                 this.volume * 100 + "%";
                },
                mute: function()
                {
                this.muted = (!this.muted) ? true : false;
                this.videoObject.muted = this.muted;
                 }
                };
                window.onload = function()
                {
                html5VideoPlayer.init();
}
```