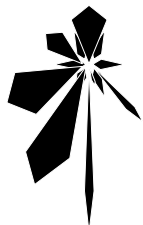


Introduction to Native Audio and 2D Animation using HTML 5 and JavaScript

Revision 1.1a
August 11, 2010

by
Aloysius Indrayanto



(C) 2010 AnemoneSoft.com

This document is multi-licensed under the Creative Commons Attribution Share-Alike (CC-BY-SA) license version 3.0 and the GNU Free Documentation License (GNU FDL) version 1.3 or later.

1. Introduction

HTML 5 is the latest standard from W3C HTML-Working-Group for structuring and presenting content on the World Wide Web (WWW). Despite the fact that the standard is still in development, some of its features are already stable enough to be used. Some of the features that may be interesting to browser-based game developers are:

- Native support for audio via the `<audio>` tag;
- Native support for video via the `<video>` tag;
- Native support for immediate-mode 2D drawing API via the `<canvas>` tag;
- Web storage (local storage and session storage).

Major browsers such as Mozilla Firefox, Google Chrome, Apple Safari, and Opera have implemented many of HTML 5's new features. However, Internet Explorer (up to version 8) still only implements a very small part of HTML 5.

Fallback mechanisms are available in case a browser does not support some of the new features. For example, the basic functionality of the `<audio>` and `<video>` tags (loading, playing, and stopping audio/video) are also supported by the `<embed>` tag. The functionality of the `<canvas>` tag can be emulated in IE by using JavaScript and IE's support for Vector Markup Language (VML). It is also possible to emulate the `<canvas>` tag using Silverlight or Flash.

This tutorial will discuss about the basic usage of `<audio>`, `<embed>`, and `<canvas>` tags. Basically, audio playback will be delegated to the `<embed>` tag if the browser in use does not support the new `<audio>` tag. In Internet Explorer, `<canvas>` tag will be supported through an emulation layer named `explorercanvas` (please refer to the reference section for more details about this emulation library).

2. Requirements

A basic knowledge in programming using JavaScript will be needed to understand the topic discussed in this tutorial. A supported browser (Internet Explorer 6.0+, Mozilla Firefox 3.5+, Google Chrome 4.0+, Apple Safari 4.0+, or Opera 10.5+) will be also needed to run the code snippets.

3. Using the <audio> Tag

Basically, audio can be played by using the new <audio> tag or using the legacy <embed> tag. The code snippet below shows various methods to play audio files. The doctype and the attribute definition of the <HTML> tag have been made simpler by the HTML 5 standard.

```
<!DOCTYPE html>
<html lang='en'>
  <head></head>
  <body>
    <div style='padding-bottom:15px; font-weight:bold;'>
      Various Methods to Play Audio in Browser</div>
    <table cellspacing='5px' cellpadding='5px' border='1'>
      <tr style='text-align:center; font-weight:bold;'>
        <td>Using &lt;audio&gt; Tag</td>
        <td>Using &lt;embed&gt; Tag</td>
      </tr>
      <tr>
        <td colspan='2'>WAVE File</td>
      </tr>
      <tr>
        <td><audio src='sound/bob-mighty-midi.wav' controls></audio></td>
        <td><embed src='sound/bob-mighty-midi.wav' autostart='false'
          controller='true' style='height:50px;'></audio></td>
      </tr>
      <tr>
        <td colspan='2'>Ogg-Vorbis File</td>
      </tr>
      <tr>
        <td><audio src='sound/bob-mighty-midi.ogg' controls></audio></td>
        <td><embed src='sound/bob-mighty-midi.ogg' autostart='false'
          controller='true' style='height:50px;'></audio></td>
      </tr>
      <tr>
        <td colspan='2'>MP3 File</td>
      </tr>
      <tr>
        <td><audio src='sound/bob-mighty-midi.mp3' controls></audio></td>
        <td><embed src='sound/bob-mighty-midi.mp3' autostart='false'
          controller='true' style='height:50px;'></audio></td>
      </tr>
      <tr>
        <td colspan='2'>MIDI File</td>
      </tr>
      <tr>
        <td><audio src='sound/bob-mighty-midi.mid' controls></audio></td>
        <td><embed src='sound/bob-mighty-midi.mid' autostart='false'
          controller='true' style='height:50px;'></audio></td>
      </tr>
    </table>
    <hr/>
    <div style='font-size:0.8em;'>
      The original MIDI file was downloaded from
      <a href='http://www.mightymidi.com'>Bob's Mighty MIDI Site</a>.
    </div>
  </body>
</html>
```

tut01a.html: Various methods to play audio in a web browser.

In browsers that support the `<audio>` tag, the result will be similar to the picture below. In browsers that do not support the `<audio>` tag, the left column of the table will be mostly empty.



Basically, to play an audio file using the `<audio>` tag, the syntax is:

```
<audio src='<url of the audio file>' controls autoplay loop/>
```

The `src` attribute must be set with the location (URL) of the audio file. The other attributes are optional:

- If the `controls` attribute is present, the browser will display control buttons and some information related to the playback.
- If the `autoplay` attribute is present, the playback will be automatically started as soon as it is ready (enough portion of the audio file has been received).
- If the `loop` attribute is present, the playback will be automatically restarted as soon as it is finished.

On the other hand, to play an audio file using the <embed> tag, the syntax is:

```
<embed src='<i>url of the audio file</i>' controller='true' autostart='true' loop='true' />
```

The src attribute must be set with the location (URL) of the audio file. The other attributes are optional:

- The controller attribute can be set so that the browser displays control buttons and some information related to the playback.
- The autostart attribute can be set so that the playback will be automatically started as soon as it is ready (enough portion of the audio file has been received).
- The loop attribute can be set so that the playback will be automatically restarted as soon as it is finished.

Some browsers may ignore (do not support) some of the above attributes.

Currently, very few audio codecs that are directly supported by the <audio> tag. A list of audio codecs that are natively supported by the fore-mentioned major browsers are shown in the table below. Some browsers may be able to support more audio codecs via additional software installation.

Browser	Ogg Vorbis	MP3	WAV
FireFox 3.5	✓		✓
Google Chrome 3.0	✓	✓	
Apple Safari 4.0		✓	✓
Opera 10.5			✓

Internet Explorer until version 8 does not supports the <audio> tag. However they do support the <embed> tag. IE will usually use the Windows Media Player (WMP) plugin to play audio files. Therefore, the audio codecs supported by IE's <embed> tag will depend on the installed WMP codecs in the system.

3.1. Selecting the Best-Supported Audio Format

A real application should select the best audio format that is supported by the browser in use. Due to different browsers will support different audio formats, it is necessary for the developer to provide multiple audio formats. Basically, to save the server's bandwidth, try to always use the smallest audio format (compressed) whenever possible. Only use the larger (but more compatible) formats if the browser does not support the compressed format.

The code snippet below demonstrate the technique to select the best-supported audio format. The available audio formats will be sorted by their level of preference. In the code snippet, the application will play a background music. The MIDI (Musical Instrument Digital Interface) format will be the smallest format for instrument-only music. It is followed by lossy-compression formats such as MP3 (MPEG-1 Audio Layer 3) and Ogg Vorbis. Finally, only if all the previous formats are unsupported, plain-uncompressed WAVE (Waveform Audio File Format) will be used.

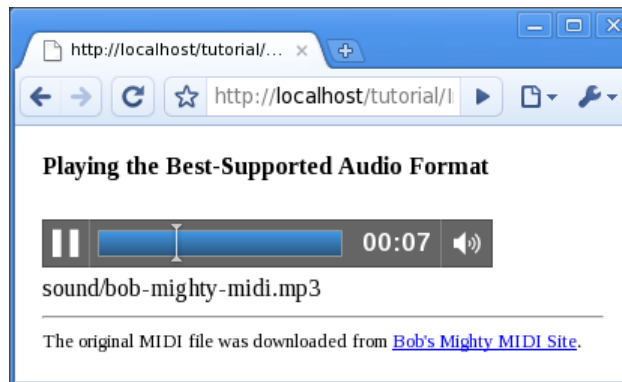
```
<!DOCTYPE html>
<html lang='en'>
  <head></head>
  <body style='padding:10px;'>
    <div style='padding-bottom:25px; font-weight:bold;'>
      Playing the Best-Supported Audio Format</div>
    <audio id='myAudioPlayer' src='' controls></audio>
    <div id='myAudioInfo'></div>
    <hr/>
    <div style='font-size:0.8em;'>
      The original MIDI file was downloaded from
      <a href='http://www.mightymidi.com'>Bob's Mighty MIDI Site</a>.
    </div>
  </body>
</html>
<script type='text/javascript'>
  var myAudioPlayer = document.getElementById('myAudioPlayer');
  var myAudioInfo   = document.getElementById('myAudioInfo' );

  var url = null;
  if(!myAudioPlayer.canPlayType) {
    if(myAudioPlayer.canPlayType('audio/mid') != '')
      url = 'sound/bob-mighty-midi.mid';
    else if(myAudioPlayer.canPlayType('audio/mpeg') != '')
      url = 'sound/bob-mighty-midi.mp3';
    else if(myAudioPlayer.canPlayType('audio/ogg') != '')
      url = 'sound/bob-mighty-midi.ogg';
    else if(myAudioPlayer.canPlayType('audio/x-wav') != '')
      url = 'sound/bob-mighty-midi.wav';
  }

  if(url) {
    myAudioInfo.innerHTML = url;
    myAudioPlayer.src = url;
  }
  else
    myAudioInfo.innerHTML =
      'Sorry, your browser does not support any of the the available audio formats!';
</script>
```

tut01b.html: Selecting the best-supported audio format.

In browsers that support the <audio> tag, the result will be similar to the picture below. In browsers that do not support the <audio> tag, a message will be shown instead.



To test if a particular audio format is supported, one can pass the MIME (Multipurpose Internet Mail Extensions) type of the audio format to the `canPlayType()` method of an audio object. The W3C standard specifies that the method shall return an empty string if the browser definitely does not support the format. If the browser supports the format, it shall return the string 'maybe' or 'probably'. The table below lists the MIME types of some audio and video formats.

Audio Format	MIME Type	Encumbered by Known Patents?
MIDI	audio/mid	should be no
AAC	audio/aac	yes
MP3	audio/mpeg	yes
Ogg Vorbis	audio/ogg	should be no
WAVE	audio/x-wav	should be no

Video Format	MIME Type	Encumbered by Known Patents?
H.264	video/h264	yes
VP8	video/vp8	should be no
Ogg Theora	video/ogg	should be no

Currently, each browser only supports few of the above formats. However, in the future browsers may support more formats. Using the `<video>` tag will be very similar to using the `<audio>` tag. Hence, it will not be discussed further in this tutorial.

3.2. Using JavaScript to Play and Stop Audio

The code snippet below shows a possible method to control audio playback using JavaScript. The script prefers to use the `<audio>` tag whenever possible. In case it is not available, the script will fallback to use the `<embed>` tag.

```
<!DOCTYPE html>
<html lang='en'>
  <head></head>
  <body style='padding:10px;'>
    <div style='padding-bottom:15px; font-weight:bold;'>
      Using JavaScript to Play and Stop Audio</div>
    <input type='button' value='Play' onclick='playAudio();'/>
    <input type='button' value='Stop' onclick='stopAudio();'/>
    <hr/>
    <div style='font-size:0.8em;'>
      The original MIDI file was downloaded from
      <a href='http://www.mightymidi.com'>Bob's Mighty MIDI Site</a>.
    </div>
  </body>
</html>
<script type='text/javascript'>
  // Test if the browser supports the 'audio' tag
  var testAudio = document.createElement('audio');
  var mid       = false;
  var mp3       = false;
  var ogg       = false;
  var wav       = false;
  var useEmbed  = false;
  if(!testAudio.canPlayType) {
    mid = (testAudio.canPlayType('audio/mid' ) != '');
    mp3 = (testAudio.canPlayType('audio/mpeg' ) != '');
    ogg = (testAudio.canPlayType('audio/ogg'  ) != '');
    wav = (testAudio.canPlayType('audio/x-wav') != '');
  }
  useEmbed = (!mid && !mp3 && !ogg && !wav);
  testAudio = null;

  // If the browser supports the 'audio' tag, create a variable
  // that will be used to store the audio object
  var objAudio = null;
  if(!useEmbed) {
    objAudio = document.createElement('audio');
    objAudio.setAttribute('autobuffer', 'autobuffer');
    objAudio.setAttribute('preload', 'auto');
  }

  // If the browser does not support the 'audio' tag, create a variable
  // that will be used to store a dummy 'div' object
  var objDiv = null;
  if(useEmbed) {
    var objDiv = document.createElement('div');
    document.body.appendChild(objDiv);
  }

  // Event handler to play the audio
  function playAudio()
  {
    // The browser does not support the 'audio' tag, use the 'embed' tag
    if(useEmbed) {
      // Here, we assume that the browser is Internet Explorer. Hence, just
      // play the smallest file (the MIDI file) because IE will most likely
      // to use the WMP plugin that usually can play MIDI files.
      objDiv.innerHTML = "<embed src='sound/bob-mighty-midi.mid' hidden='true' "
        + "autostart='true' loop='false'>";
      return;
    }

    // The browser supports the 'audio' tag :)
    if(mid) objAudio.src = 'sound/bob-mighty-midi.mid';
    else if(mp3) objAudio.src = 'sound/bob-mighty-midi.mp3';
    else if(ogg) objAudio.src = 'sound/bob-mighty-midi.ogg';
    else if(wav) objAudio.src = 'sound/bob-mighty-midi.wav';
  }
</script>
```

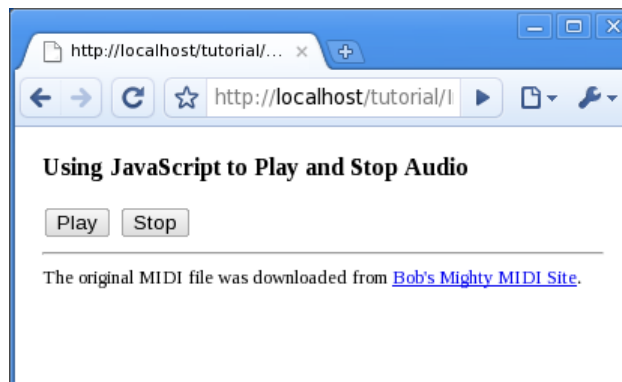
```
    if(objAudio.load) objAudio.load();
    objAudio.play();
}

// Event handler to stop the audio
function stopAudio()
{
    // The browser does not support the 'audio' tag, use the 'embed' tag
    if(useEmbed) {
        objDiv.innerHTML = '';
        return;
    }

    // The browser supports the 'audio' tag :)
    objAudio.pause();
    objAudio.currentTime = 0;
}
</script>
```

tut01c.html: Using JavaScript to play and stop audio.

The result will be similar to the picture below.



Explanation:

- The play and stop buttons are hooked to onclick event handlers playAudio() and stopAudio() respectively.
- After the document is fully loaded, the onload event will be fired. The event handler for this event will first check if the browser support the <audio> tag and if it supports at least one of the available file formats (MIDI, MP3, Ogg Vorbis, or WAVE). If the browser does not support the <audio> tag or any of the available file formats, the script will fallback to use the <embed> tag:

```
var testAudio = document.createElement('audio');
var mid      = false;
...
var useEmbed = false;
if(!testAudio.canPlayType) {
    mid = (testAudio.canPlayType('audio/mid') != '');
    ...
}
useEmbed = (!mid && !mp3 && !ogg && !wav);
testAudio = null;
```

- If the `<audio>` tag will be used, the script will then initialize the audio object and try to enable auto-buffering/pre-loading:

```
var objAudio = null;
if(!useEmbed) {
  objAudio = document.createElement('audio');
  objAudio.setAttribute('autobuffer', 'autobuffer');
  objAudio.setAttribute('preload', 'auto');
}
```

Otherwise, the script will initialize a dummy 'div' object to be used to dynamically store the `<embed>` tag:

```
var objDiv = null;
if(useEmbed) {
  var objDiv = document.createElement('div');
  document.body.appendChild(objDiv);
}
```

- If the `<audio>` tag is used, the `playAudio()` method will set the source URL of the audio object, load the audio, and finally play it:

```
if(mid) objAudio.src = 'sound/bob-mighty-midi.mid';
else if(mp3) objAudio.src = 'sound/bob-mighty-midi.mp3';
...
if(objAudio.load) objAudio.load();
objAudio.play();
```

Otherwise, the script will dynamically create a new `<embed>` tag inside the dummy 'div' that points to the audio file's URL:

```
objDiv.innerHTML = "<embed src='sound/bob-mighty-midi.mid' hidden='true' "
+ "autostart='true' loop='false'>"
```

The script assumes that only Internet Explorer will need to use the `<embed>` tag. Hence, it just plays the smallest file (the MIDI file) because IE will most likely to use the WMP plugin that usually can play MIDI files.

- If the `<audio>` tag is used, the `stopAudio()` method will pause the playback and rewind the audio stream (the `<audio>` tag does not provide any method to stop playback):

```
objAudio.pause();
objAudio.currentTime = 0;
```

Otherwise, the script will just remove (delete) the `<embed>` tag from the dummy 'div' to stop the playback:

```
objDiv.innerHTML = '';
```

Note that, using the `<embed>` tag in some browsers may cause undesired effect. For example, Konqueror 4.3.2 with WebKit rendering engine may actually run an independent media player application instead of embedding the player in the browser.

4. Using the <canvas> Tag

The <canvas> tag is not yet supported by Internet Explorer up to version 8. However, a JavaScript emulation library is available. The latest revision of `explorercanvas` (revision 73) can be obtained from its Subversion (SVN) repository. It implements almost all the important functionality of the <canvas> tag. However, some issues do exist:

- The doctype will need to be set to HTML 5 for IE to correctly draw the canvas.
- The method `clearRect()` will always clear the whole canvas (not just the specified rectangle).
- The method `createPattern()` only supports the mode `repeat`; other modes (`repeat-x`, `repeat-y`, and `no-repeat`) are not supported.
- Non uniform scaling does not correctly scale strokes.
- Shadow rendering is not implemented (although the corresponding attributes do exist).
- The methods `createRadialGradient()`, `clip()`, `arcTo()`, and `isPointInPath()` are not implemented.
- Painting mode (via the `globalCompositeOperation` attribute) is not implemented.
- Converting to image (via the `toDataURL()` method) is not implemented.
- Pixel manipulation (direct access to the image data) is not implemented.
- Maybe some other undocumented/unknown incompatibilities.

To use this library in IE, simply add a one-line code to load the script to the <head></head> section of the HTML document, such as:

```
<!--[if IE]><script type='text/javascript' src='excanvas-c73p.js'></script><![endif]-->
```

To ensure that the library has finished its initialization, all canvas operations must be done only after the `onload` event is fired. Note that if the canvas object is generated dynamically using the `document.createElement()` method, it will be necessary to manually initialize the canvas by calling the `G_vmlCanvasManager.initElement()` method before attempting to use the newly created object as a canvas object:

```
var myCanvas = document.createElement('canvas');  
if(typeof(G_vmlCanvasManager) != 'undefined') G_vmlCanvasManager.initElement(myCanvas);
```

The `if` branch is needed so that the code will not produce error on browsers that natively support canvas. The code snippet below demonstrate some features of canvas. Try to play and experiment with the code snippet by modifying the parameters of the method calls.

```
<!DOCTYPE html>
<html lang='en'>
  <head>
    <!--[if IE]><script type='text/javascript' src='excanvas-c73p.js'></script><![endif]-->
  </head>
  <body style='padding:10px;' onload='doDraw();'>
    <canvas id='myCanvas' width='400' height='200'></canvas>
    <input type='button' value='Refresh Canvas' onclick='doDraw();'
      style='vertical-align:top;' />
    <hr />
    <div style='font-size:0.8em;'>
      The original chicken image was downloaded from
      <a href='http://www.sitevip.net/gifs/chicken'>
        http://www.sitevip.net/gifs/chicken</a>.
    </div>
    <img id='imgChickenBaby' src='image/chicken-baby.gif' />
  </body>
</html>
<script type='text/javascript'>
  function doDraw()
  {
    // Get access to the canvas
    var myCanvas = document.getElementById('myCanvas');
    var vCtx      = myCanvas.getContext('2d');

    // Set the canvas background and border
    myCanvas.style.background = '#000000';
    myCanvas.style.border     = '3px solid #7F7F7F';

    // Clear the canvas
    vCtx.clearRect(0, 0, myCanvas.width, myCanvas.height);

    // Draw rectangles
    vCtx.fillStyle = '#00FF00';
    vCtx.strokeStyle = '#FF0000';
    vCtx.lineWidth = 2;

    vCtx.fillRect (10, 10, 50, 50);
    vCtx.strokeRect(30, 30, 50, 50);

    // Draw a path
    vCtx.beginPath();
    vCtx.moveTo(100, 25);
    vCtx.lineTo(175, 50);
    vCtx.lineTo(125, 65);
    vCtx.closePath();

    vCtx.strokeStyle = '#0000FF';
    vCtx.stroke();

    // Draw rectangles using gradient
    var vGrad1 = vCtx.createLinearGradient(200, 0, 250, 0);
    vGrad1.addColorStop(0.0, '#FF0000');
    vGrad1.addColorStop(0.5, '#00FF00');
    vGrad1.addColorStop(1.0, '#0000FF');
    vCtx.fillStyle = vGrad1;
    vCtx.fillRect(200, 10, 50, 50);

    var vGrad2 = vCtx.createLinearGradient(260, 10, 310, 60);
    vGrad2.addColorStop(0.0, '#FF0000');
    vGrad2.addColorStop(0.5, '#00FF00');
    vGrad2.addColorStop(1.0, '#0000FF');
    vCtx.fillStyle = vGrad2;
    vCtx.fillRect(260, 10, 50, 50);

    // Draw a circle with glow (white shadow)
    vCtx.shadowBlur = 25;
    vCtx.shadowColor = '#FFFFFF';
```

```
vCtx.shadowOffsetX = 0;
vCtx.shadowOffsetY = 0;
vCtx.beginPath();
vCtx.arc(350, 35, 25, 0, Math.PI * 2, false);
vCtx.fill();

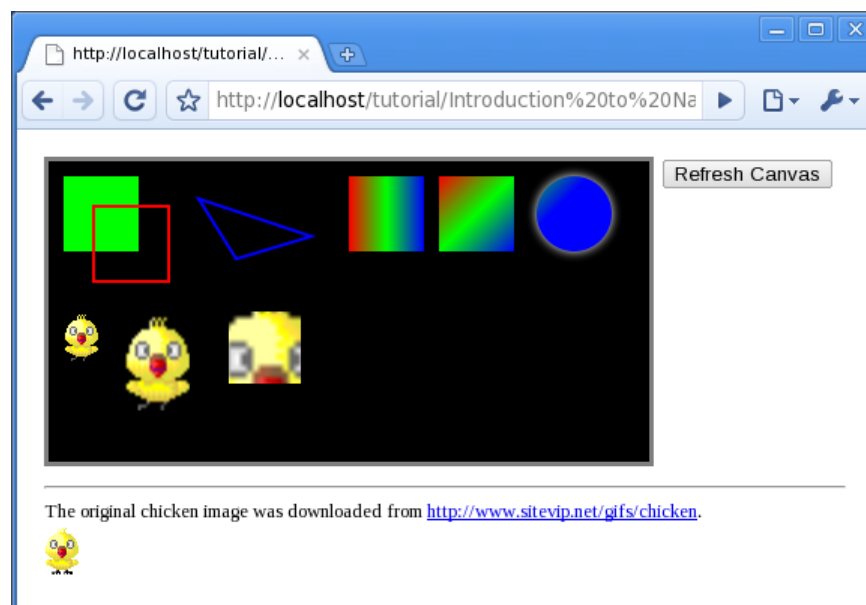
vCtx.shadowColor = 'transparent black';

// Draw images
var imgChickenBaby = document.getElementById('imgChickenBaby');

vCtx.drawImage(imgChickenBaby, 10, 100, 44, 66);
vCtx.drawImage(imgChickenBaby, 50, 100, 44, 66);
vCtx.drawImage(imgChickenBaby, 5, 5, 12, 12, 120, 100, 48, 48);
}
</script>
```

tut02a.html: Using the <canvas> tag.

The result will be similar to the picture below.



Explanation:

- First, we need to define the canvas style (background/clear color and border) by using standard CSS such as:

```
myCanvas.style.background = '#000000';
myCanvas.style.border = '3px solid #7F7F7F';
```

Next, to start using the canvas, we need to get access to its 2D context:

```
var vCtx = myCanvas.getContext('2d');
```

The whole canvas can be cleared by using the statement:

```
vCtx.clearRect(0, 0, myCanvas.width, myCanvas.height);
```

The W3C standard states that the method `clearRect()` takes four parameters: the starting coordinate (x, y) and the size (w, h) of the area to be cleared. However `explorercanvas` ignores the parameters and always clear the entire canvas.

- To draw a rectangle, the methods `fillRect()` and `strokeRect()` can be used. They take four parameters: the starting coordinate (x, y) and the size (w, h) of the rectangle. The attributes `fillStyle` and `strokeStyle` can be used to set the colors of the area and line.
- Drawing a path can be done by a sequence of method calls, such as:

<pre>vCtx.beginPath(); ... vCtx.closePath(); vCtx.fill();</pre>	<pre>vCtx.beginPath(); ... vCtx.closePath(); vCtx.stroke();</pre>	<pre>vCtx.beginPath(); ... vCtx.stroke();</pre>
Filled Path	Stroke Path (Closed)	Stroke Path (Open)

The ... in the sequence can be filled with calls to path-related drawing methods. Some examples of method calls (with their expected parameters) are:

```
vCtx.moveTo(x, y);
vCtx.lineTo(x, y);
vCtx.arc(x, y, radius, start_angle, end_angle, anticlockwise);
```

- Gradient can be specified and used by using code such as:

```
var grad = vCtx.createLinearGradient(x1, y1, x2, y2);
grad.addColorStop(offset1, color1);
grad.addColorStop(offset2, color2);
grad.addColorStop(offset3, color3);
...
vCtx.fillStyle = grad;
```

Note that an offset specify a relative position (0.0 to 1.0) for the given color in the gradient (specify a color key).

- Drawing (putting) image on the canvas can be done by using the `drawImage()` method.

The code below shows an example of how to use all the three overloads of the method:

```
vCtx.drawImage(image, dx, dy ); // Normal
vCtx.drawImage(image, dx, dy, dw, dh); // With scaling
vCtx.drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh); // With clipping & scaling
```

If the `image` object contains an animated image (for example animated GIF), some browsers may always render the first frame of the image. The `image` object can come from HTML's `` tag or created dynamically using JavaScript.

4.1. Simple Animation using the `<canvas>` Tag

Animation can be achieved by combining the `<canvas>` tag and the `setTimeout()` method from the window object. The code snippet below demonstrate this technique. The result will be similar to the picture shown next.

```
<!DOCTYPE html>
<html lang='en'>
  <head>
    <!--[if IE]><script type='text/javascript' src='excanvas-c73p.js'></script><![endif]-->
  </head>
  <body style='padding:10px;' onload='doDraw();'>
    <canvas id='myCanvas' width='400' height='200'></canvas>
    <hr/>
    <div style='font-size:0.8em;'>
      The original chicken image was downloaded from
      <a href='http://www.sitevip.net/gifs/chicken'>
        http://www.sitevip.net/gifs/chicken</a>.
    </div>
    <img id='imgChickenBaby' src='image/chicken-baby.gif'/>
  </body>
</html>
<script type='text/javascript'>
  var rotCur = 0;
  var rotInc = Math.PI * 0.02;
  var rotMax = Math.PI * 2
  function doDraw()
  {
    // Get access to the image
    var imgChickenBaby = document.getElementById('imgChickenBaby');

    // Get access to the canvas
    var myCanvas = document.getElementById('myCanvas');
    var ctrX     = myCanvas.width * 0.5;
    var ctrY     = myCanvas.height * 0.5;
    var vCtx     = myCanvas.getContext('2d');

    // Set the canvas background and border
    myCanvas.style.background = '#000000';
    myCanvas.style.border    = '3px solid #7F7F7F';

    // Clear the canvas
    vCtx.clearRect(0, 0, myCanvas.width, myCanvas.height);

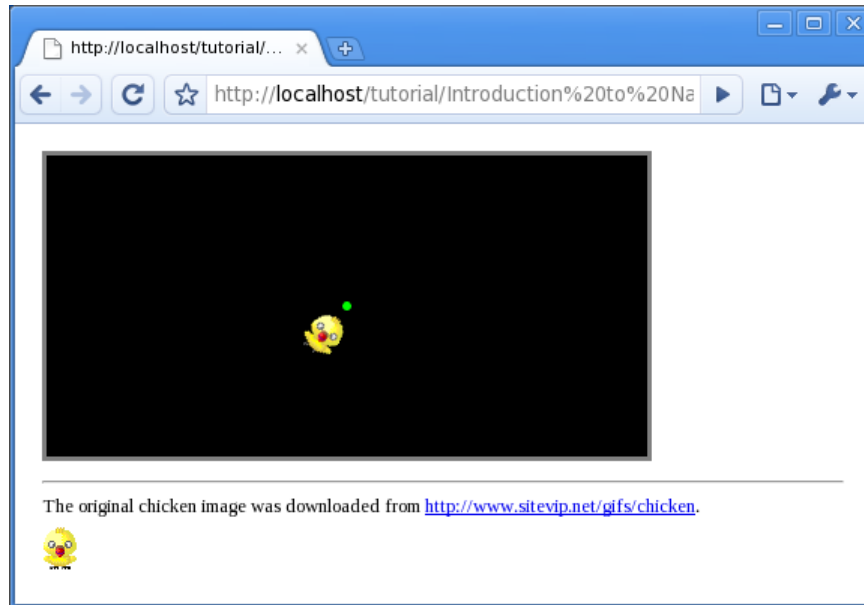
    // Draw a circle at the center of the canvas
    vCtx.fillStyle = '#00FF00';
    vCtx.strokeStyle = '#00FF00';
    vCtx.beginPath();
    vCtx.arc(ctrX, ctrY, 3, 0, Math.PI * 2, true);
    vCtx.fill();

    // Draw a transformed image
    vCtx.save();
    vCtx.translate(ctrX, ctrY);
    vCtx.rotate(rotCur);
    vCtx.translate(-imgChickenBaby.width * 0.5, 5);
    vCtx.drawImage(imgChickenBaby, 0, 0);
    vCtx.restore();

    // Update the rotation
    rotCur += rotInc;
    if(rotCur > rotMax) rotCur = 0;

    // Set timeout for the next frame
    setTimeout(arguments.callee, 50);
  }
</script>
```

tut02b.html: Simple animation using the <canvas> tag.



Explanation:

- The heart of the animation lies in the call of the methods `save()`, `restore()`, `translate()`, and `rotate()`:
 - first, the method `save()` is called to save the current state of the canvas;
 - next, a sequence of transformations (translation and rotation) is applied;
 - then, the image is drawn;
 - finally, the method `restore()` is called to restore the state of the canvas;
- Transformation is applied in reverse order from the positions where they appear in the code. Therefore a sequence of commands:

```
vCtx.translate(ctrX, ctrY);  
vCtx.rotate(rotCur);  
vCtx.translate(-imgChickenBaby.width * 0.5, 5);  
vCtx.drawImage(imgChickenBaby, 0, 0);
```

will translate the image-to-be-drawn (`imgChickenBaby`) by `(-width*0.5, 5)`, then rotate it by `rotCur`, before finally translating it again by `(ctrX, ctrY)`.

4.2. Transparency in the `<canvas>` Tag

Transparency is fully supported in canvas. The `globalAlpha` attribute from the canvas' 2D context object can be modified to specify the wanted transparency. The code snippet below shows the needed modification to the previous code snippet to draw multiple chicken images, each with different transparency level. The result will be similar to the picture shown next.

```
...
...
...

// Draw a circle at the center of the canvas
vCtx.fillStyle = '#00FF00';
vCtx.strokeStyle = '#00FF00';
vCtx.beginPath();
vCtx.arc(ctrX, ctrY, 3, 0, Math.PI * 2, true);
vCtx.fill();

// Draw transformed images
vCtx.save(); // Save the canvas state

vCtx.translate(ctrX, ctrY); // Translate to the center of the canvas
vCtx.rotate(rotCur); // Rotate the image
vCtx.translate(-imgChickenBaby.width * 0.5, 0); // Center the image horizontally

vCtx.globalAlpha = 0.25; // Set transparency
vCtx.translate(30, 35); // Add some offset
vCtx.drawImage(imgChickenBaby, 0, 0); // Draw the image

vCtx.globalAlpha = 0.5; // Set transparency
vCtx.translate(-10, -10); // Adjust the offset
vCtx.drawImage(imgChickenBaby, 0, 0); // Draw the image

vCtx.globalAlpha = 0.75; // set transparency
vCtx.translate(-10, -10); // Adjust the offset
vCtx.drawImage(imgChickenBaby, 0, 0); // Draw the image

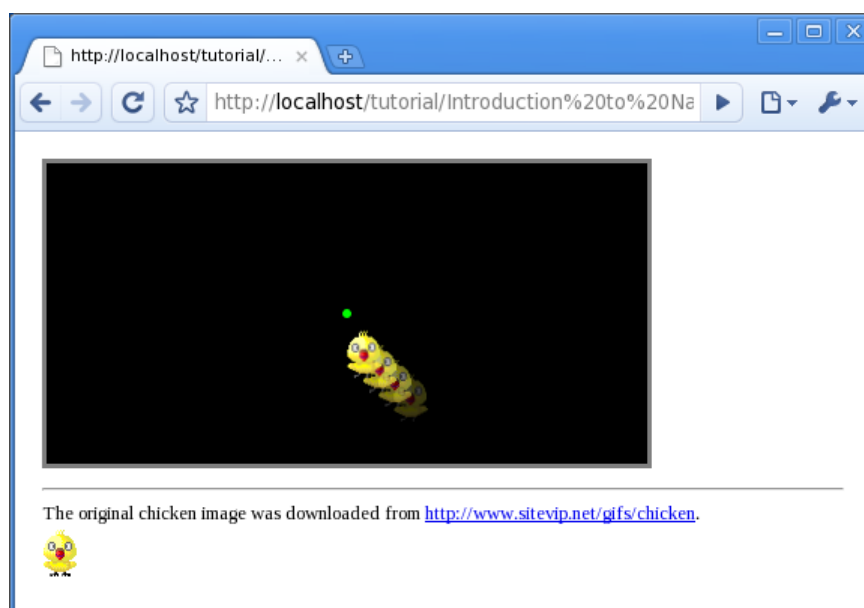
vCtx.globalAlpha = 1.0; // Set transparency
vCtx.translate(-10, -10); // Adjust the offset
vCtx.drawImage(imgChickenBaby, 0, 0); // Draw the image

vCtx.restore(); // Restore the canvas state

// Update the rotation
rotCur += rotInc;
if(rotCur > rotMax) rotCur = 0;

...
...
...
```

tut02c.html: Transparency in animation using the <canvas> tag.



Explanation:

- The modification is basically done to the statements for drawing the image. Instead of only one image, now four images are drawn.
- The first image has 0.25 alpha (25% opacity) and it is transformed with this sequence of transformation:

$$T(\text{ctrX}, \text{ctrY}) \cdot R(\text{rotCur}) \cdot T(-\text{width} \cdot 0.5, 0) \cdot T(30, 35) = \\ T(\text{ctrX}, \text{ctrY}) \cdot R(\text{rotCur}) \cdot T(-\text{width} \cdot 0.5 + 30, 35)$$

Remember that the last transformation will be applied first.

- The second image has 0.5 alpha (50% opacity) and it is transformed by this simplified sequence of transformation:

$$T(\text{ctrX}, \text{ctrY}) \cdot R(\text{rotCur}) \cdot T(-\text{width} \cdot 0.5 + 20, 25)$$

- The third image has 0.75 alpha (75% opacity) and it is transformed by this simplified sequence of transformation:

$$T(\text{ctrX}, \text{ctrY}) \cdot R(\text{rotCur}) \cdot T(-\text{width} \cdot 0.5 + 10, 15)$$

- The fourth image has 1.0 alpha (100% opacity) and it is transformed by this simplified sequence of transformation:

$$T(\text{ctrX}, \text{ctrY}) \cdot R(\text{rotCur}) \cdot T(-\text{width} \cdot 0.5, 5)$$

4.3. Rendering Text using the <canvas> Tag

Text can be drawn in two modes, outline and solid, using the method `strokeText()` and `fillText()` from the 2D context. Solid text can be filled with solid color or gradient. The code snippet below demonstrate this feature.

```
<!DOCTYPE html>
<html lang='en'>
  <head>
    <!--[if IE]><script type='text/javascript' src='excanvas-c73p.js'></script><![endif]-->
  </head>
  <body style='padding:10px;' onload='doDraw();'>
    <canvas id='myCanvas' width='400' height='200'></canvas>
  </body>
</html>
<script type='text/javascript'>
  function doDraw()
  {
    // Get access to the canvas
    var myCanvas = document.getElementById('myCanvas');
    var vCtx     = myCanvas.getContext('2d');

    // Set the canvas background and border
    myCanvas.style.background = '#000000';
    myCanvas.style.border    = '3px solid #7F7F7F';
```

```
// Clear the canvas
vCtx.clearRect(0, 0, myCanvas.width, myCanvas.height);

// Set text parameters
vCtx.font = 'bold 50px "courier new"';
vCtx.textAlign = 'left';
vCtx.textBaseline = 'top';

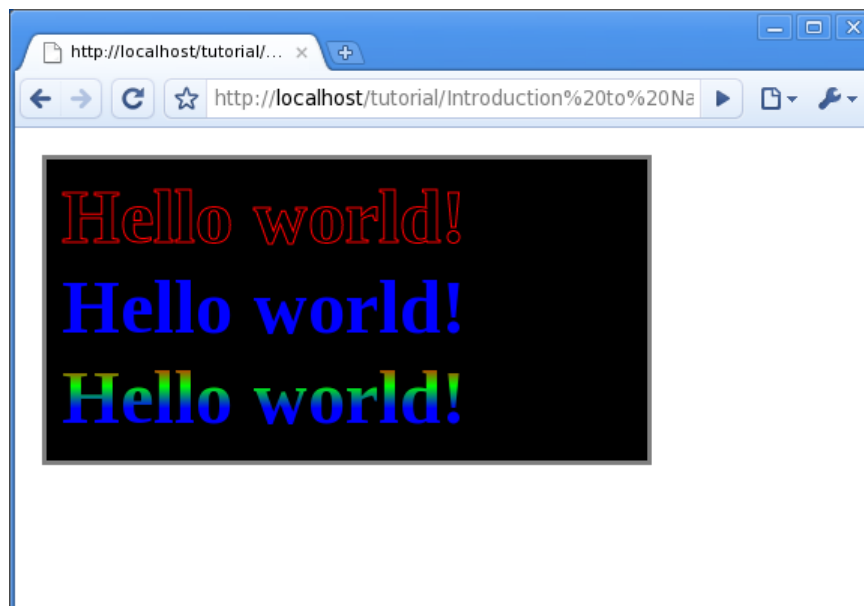
// Draw outline text
vCtx.lineWidth = 1;
vCtx.strokeStyle = '#FF0000';
vCtx.strokeText('Hello world!', 10, 10);

// Draw solid text
vCtx.fillStyle = '#0000FF';
vCtx.fillText('Hello world!', 10, 70);

// Draw gradient text
var vGrad1 = vCtx.createLinearGradient(0, 135, 0, 165);
vGrad1.addColorStop(0.0, '#FF0000');
vGrad1.addColorStop(0.5, '#00FF00');
vGrad1.addColorStop(1.0, '#0000FF');
vCtx.fillStyle = vGrad1;
vCtx.fillText('Hello world!', 10, 130);
}
</script>
```

tut02d.html: Rendering text using the <canvas> tag.

The result will be similar to the picture shown below.



Explanation:

- Basically, these attributes affect how the text will be rendered:
 - font which specifies the font to be used to render the text. Example: '20px arial' and 'bold 50px "courier new"'. It is specified like a CSS style.
 - textAlign which specifies the horizontal alignment of the rendered text relative to the specified coordinate. Valid values are:

Value	Description
left	The text is left-aligned.
right	The text is right-aligned.
center	The text is centered.
start	The text is aligned at the normal start of the line (left-aligned for left-to-right locales, right-aligned for right-to-left locales).
end	The text is aligned at the normal end of the line (right-aligned for left-to-right locales, left-aligned for right-to-left locales).

- `textBaseline` which defines the baseline of the rendered text relative to the specified coordinate. Some valid values are:

Value	Description
top	The text baseline is the top of the em square.
middle	The text baseline is the middle of the em square.
bottom	The text baseline is the bottom of the bounding box.

The picture below demonstrates various baselines settings as standardized by W3C. The picture is taken from the WHATWG (Web Hypertext Application Technology Working Group) website (<http://www.whatwg.org>).



Some of those above settings are not yet implemented by current browser implementations. The canvas emulation library, `explorercanvas`, implements even fewer settings.

- The `strokeText()` and `fillText()` methods both take three parameters, the text to be drawn, the X coordinate, and the Y coordinate.
- The `measureText()` method takes a text and return the width of the text in pixels using the current font style and settings.

Try to play and experiment with the code snippet by modifying the parameters of the method calls.

5. Combining the <audio> and <canvas> Tags

The code snippet below demonstrate the combination of the <audio> (or <embed>, if the former is not supported) and <canvas> tags to create a bouncing ball animation with sound.

```
<!DOCTYPE html>
<html lang='en'>
  <head>
    <!--[if IE]><script type='text/javascript' src='excanvas-c73p.js'></script><![endif]-->
  </head>
  <body style='padding:10px;' onload='doDraw();'>
    <canvas id='myCanvas' width='400' height='200'></canvas>
    <hr/>
    <div style='font-size:0.8em;'>
      The original audio file was downloaded from
      <a href='http://www.partnersinrhyme.com'>Royalty Free Music and Sound Effects</a>.
    </div>
  </body>
</html>
<script type='text/javascript'>
  // Test if the browser supports the 'audio' tag
  var testAudio = document.createElement('audio');
  var mp3       = false;
  var ogg       = false;
  var wav       = false;
  var useEmbed  = false;
  if(!testAudio.canPlayType) {
    mp3 = (testAudio.canPlayType('audio/mpeg') != '');
    ogg = (testAudio.canPlayType('audio/ogg') != '');
    wav = (testAudio.canPlayType('audio/x-wav') != '');
  }
  useEmbed = (!mp3 && !ogg && !wav);
  testAudio = null;

  // If the browser supports the 'audio' tag, create a variable
  // that will be used to store the audio object
  var objAudio = null;
  if(!useEmbed) {
    objAudio = document.createElement('audio');
    objAudio.setAttribute('autobuffer', 'autobuffer');
    objAudio.setAttribute('preload', 'auto');
  }

  // If the browser does not support the 'audio' tag, create a variable
  // that will be used to store a dummy 'div' object
  var objDiv = null;
  if(useEmbed) {
    var objDiv = document.createElement('div');
    document.body.appendChild(objDiv);
  }

  // Event handler to play the audio
  function playAudio()
  {
    // The browser does not support the 'audio' tag, use the 'embed' tag
    if(useEmbed) {
      objDiv.innerHTML =
        "<embed src='sound/cling.wav' hidden='true' autostart='true' loop='false'>";
      return;
    }

    // The browser supports the 'audio' tag :)
    if(mp3) objAudio.src = 'sound/cling.mp3';
    else if(ogg) objAudio.src = 'sound/cling.ogg';
  }
</script>
```

```
    else if(wav) objAudio.src = 'sound/cling.wav';

    if(objAudio.load) objAudio.load();
    objAudio.play();
}

// Ball animation
var rad = 8;
var curX = rad;
var curY = 100;
var spdX = 7;
var spdY = 7;

function doDraw()
{
    // Get access to the canvas
    var myCanvas = document.getElementById('myCanvas');
    var vCtx = myCanvas.getContext('2d');
    var vWidth = myCanvas.width;
    var vHeight = myCanvas.height;

    // Set the canvas background and border
    myCanvas.style.background = '#000000';
    myCanvas.style.border = '3px solid #7F7F7F';

    // Clear the canvas
    vCtx.clearRect(0, 0, myCanvas.width, myCanvas.height);

    // Draw a ball (composed of some circles) at the current coordinate
    vCtx.fillStyle = '#004000';
    vCtx.beginPath();
    vCtx.arc(curX, curY, rad, 0, Math.PI * 2, true);
    vCtx.fill();

    vCtx.fillStyle = '#008000';
    vCtx.beginPath();
    vCtx.arc(curX, curY, rad * 0.75, 0, Math.PI * 2, true);
    vCtx.fill();

    vCtx.fillStyle = '#00CF00';
    vCtx.beginPath();
    vCtx.arc(curX, curY, rad * 0.5, 0, Math.PI * 2, true);
    vCtx.fill();

    vCtx.fillStyle = '#00FF00';
    vCtx.beginPath();
    vCtx.arc(curX, curY, rad * 0.25, 0, Math.PI * 2, true);
    vCtx.fill();

    // Flag to indicate if a sound needs to be played
    var playSound = false;

    // Update the coordinate
    curX += spdX;
    if(curX <= 0) {
        curX = rad;
        spdX = -spdX;
        playSound = true;
    }
    if(curX >= vWidth - rad) {
        curX = vWidth - rad;
        spdX = -spdX;
        playSound = true;
    }

    curY += spdY;
    if(curY <= 0) {
        curY = rad;
        spdY = -spdY;
        playSound = true;
    }
}
```

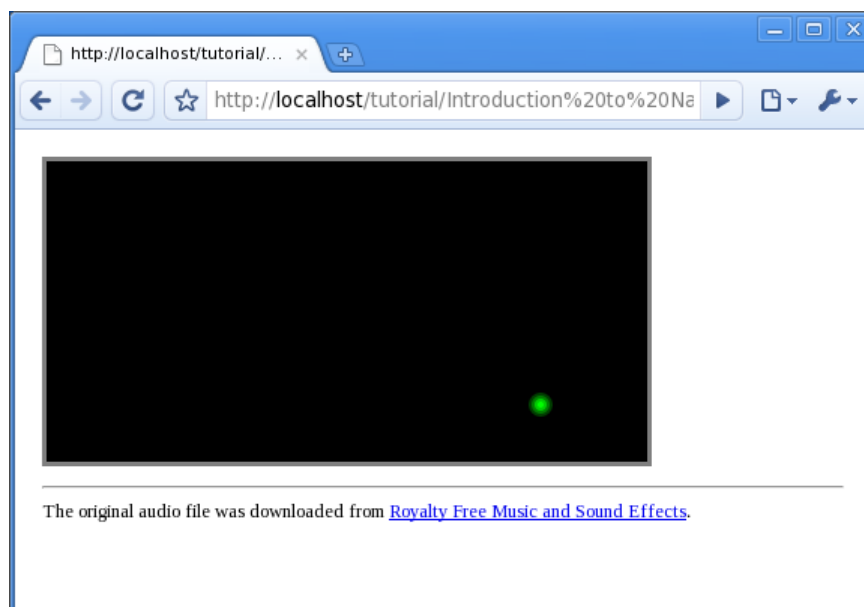
```
    if(curY >= vHeight - rad) {
        curY      = vHeight - rad;
        spdY      = -spdY;
        playSound = true;
    }

    // Play the sound (if needed)
    if(playSound) playAudio();

    // Set timeout for the next frame
    setTimeout(arguments.callee, 50);
}
</script>
```

tut03.html: Combining the <audio> and <canvas> tags.

The result will be similar to the picture shown below.



We are sure that you will be able to understand the code yourself. Try to play and experiment with the code snippet.

References

<http://en.wikipedia.org/wiki/HTML5>, June 16, 2010

<http://www.khronos.org/webgl>, June 20, 2010

<http://html5doctor.com/native-audio-in-the-browser>, June 20, 2010

https://developer.mozilla.org/en/drawing_text_using_a_canvas, July 08, 2010

<http://www.whatwg.org>, July 08, 2010

<http://me.eae.net/stuff/iecanvas>, June 16, 2010

<http://code.google.com/p/explorercanvas>, June 16, 2010

<http://www.sencha.com/playpen/tm/excanvas-patch>, June 20, 2010

http://en.wikipedia.org/wiki/Microsoft_Silverlight, July 05, 2010

<http://www.picnet.com.au/blogs/Guido/post/2010/03/15/>

[Google-Explorercanvas-\(excanvas\)-for-IE-Silverlight-vs-VML.aspx](#), July 05, 2010

<http://flashcanvas.net>, July 05, 2010

http://en.wikipedia.org/wiki/Musical_Instrument_Digital_Interface, July 07, 2010

<http://en.wikipedia.org/wiki/Mp3>, July 07, 2010

http://en.wikipedia.org/wiki/Ogg_vorbis, July 07, 2010

<http://en.wikipedia.org/wiki/WAV>, July 07, 2010

<http://en.wikipedia.org/wiki/MIME>, July 07, 2010

http://en.wikipedia.org/wiki/Apache_Subversion, July 07, 2010