Mobile Hybrid Applications

www.LifeMichael.com

HTML5

Introduction

What is HTML 5.0?

- HTML 5.0 is the new coming specification that aims at replacing HTML 4.01 and XHTML 1.0.
- The main target of HTML 5.0 is to reduce the need for proprietary plug-in-based rich internet application technologies, such as JavaFX, Silverlight and Adobe Flash.

The Standardization Process

- The work on HTML 5.0 has started in June 2004. As of June 2010 it is still in a working draft state at W3C.
- The development of HTML 5.0 is lead by Ian Hickson from Google.
- Parts of HTML 5.0 are already supported by some of the web browsers.

Markup Language

- HTML 5.0 introduces new elements and new attributes that aims at assisting with the development of web applications in accordance with today standards.
- Some of the new tags are semantic only, such as the `<nav>` and the `<footer>` tags that replace `<div>`.
- Some of the new tags, such as the `<audio>` and the `<video>` tags, provide new functionality.

Markup Language

- Some of the HTML 4.01 deprecated tags, such as `<font>` and `<center>` were dropped.

Application Programming Interface

- In addition to the new markup elements, HTML 5.0 specifies new scripting application programming interfaces (APIs).

  Canvas 2D Graphics

  Browser History Management

  Offline Storage Database

  Media Playback

  Drag & Drop

Geo Location

Introduction

- The Geo Location API allows the users to share their location with web sites they trust.
- HTML 5.0 allows us to write JavaScript code that finds out the latitude and the longitude.

The `navigator.geolocation` Property

- The `geolocation` new property on the global `navigator` object provides us with the capability to get the location information.

The `getCurrentPosition` Function

- Calling the `getCurrentPositioneolocation` function on the `navigator.geolocation` property we can pass over a callback function.
- The callback function will be called in case the Geo Location API is supported and the user approves it to use his geo location data.

```
...
navigator.geolocation.getCurrentPosition(myfunc,myerrorfunc);
...
```

The Location Callback Function

```
...
function myfunc(position)
{
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    var altitude = position.coords.altitude;
    var accuracy = position.coords.accuracy;  //meters
    var altitudeAccuracy = position.coords.altitudeAccuracy;  //meters
    var heading = position.coords.heading;  //degrees clockwise from north
    var speed = position.coords.speed;  //speed in meters/seconds
}
...
```

usually latitude, longitude and accuracy are the only ones that get support

The Error Callback Function

```
...
function myerrorfunc(errorobject)
{
    var message = errorobject.message;
    var code = errorobject.code;
    ...
}
...
```

The possible error code values are:

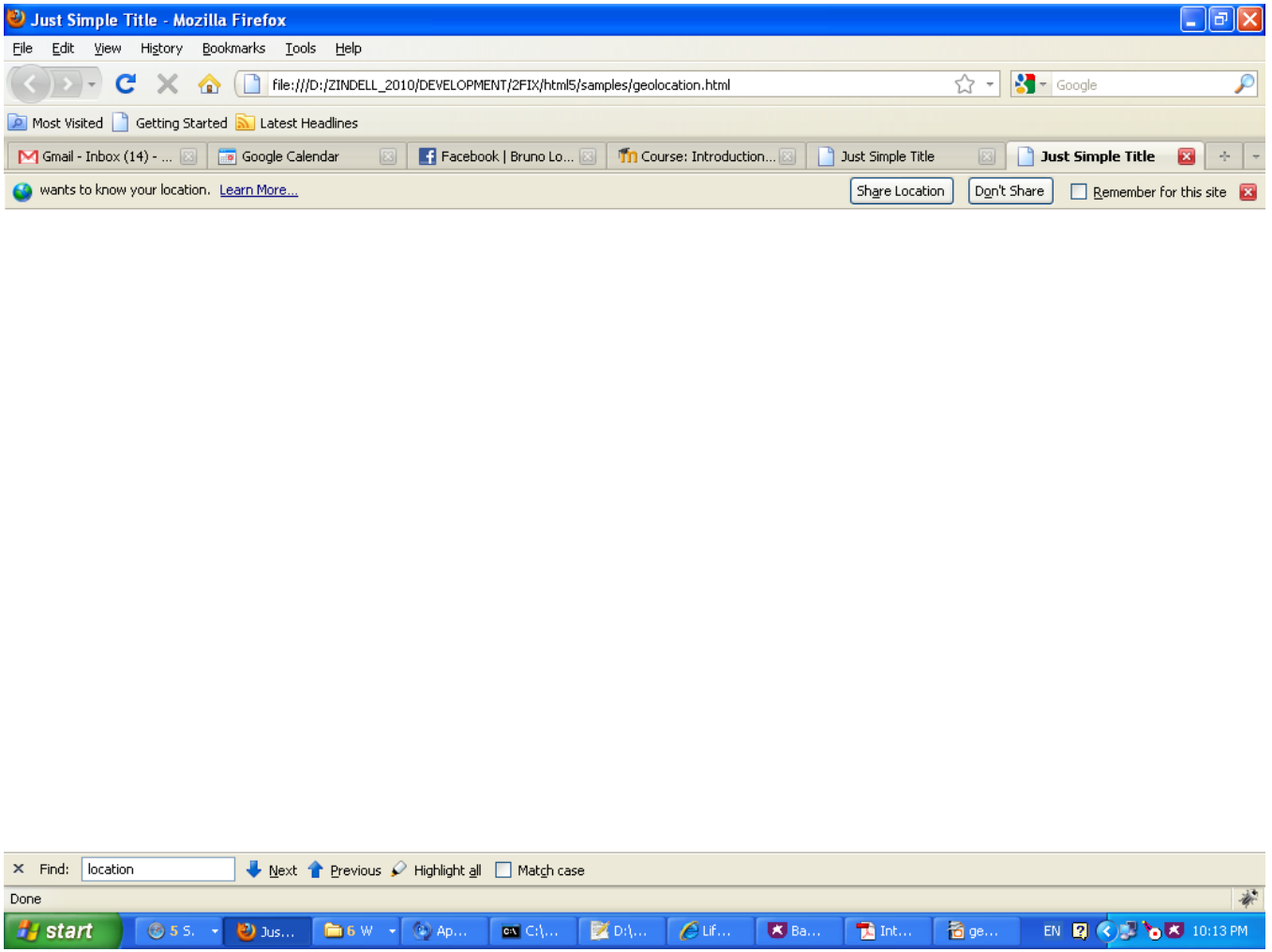
```
PERMISSION_DENIED
POSITION_UNAVAILABLE
TIMEOUT
UNKNOWN_ERROR
```

Geo Location Code Sample

```
<html>
  <head>
    <title>Just Simple Title</title>
    <script language="javascript">
    function myfunc(ob)
    {
      alert("latitude="+ob.coords.latitude+" longitude="+ob.coords.longitude);
    }
    function errfunc(ob)
    {
      alert(ob.message);
    }
    if(window.navigator.geolocation)
    {
      window.navigator.geolocation.getCurrentPosition(myfunc,errfunc);
    }
    else
    {
      alert("geolocation is not supported");
    }
    </script>
    </head>
    <body>
    </body>
</html>
```

the support for this api doesn't exist on every browser and those that do support this api still don't support it 100%.

Geo Location Code Sample



Offline Storage

Introduction

- The HTML 5.0 specification supports a well structured offline storage solution.
- There are different types of offline storages:

  Session Storage

  Local Storage

Session Storage

- The session storage extends the capabilities we get when using cookies.
- Unlike the cookies mechanism that limits us for storing up to 4kilobytes of data the session storage allows us much more space.
- Unlike the cookies mechanism, the session data isn't sent automatically to the server every HTTP request. The developer can choose the exact key value pairs to be sent.

Session Storage

- Unlike cookies, session storage is tied to the browser tab/window. Each tab/window maintains its own session information.

Session Storage

```
<html>
    <head>
        <title>session storage page 1</title>
    </head>
    <body>
        <h2>page 1</h2>
        <script language="javascript">
            sessionStorage.setItem('company', 'Zindell Technologies');
        </script>
        The 'company' (id) and 'Zindell Technologies' (value) were set as a key
        value pair in the session storage mechanism.
        <p>
        <a href="session_storage_2.html">next</a>
    </body>
</html>
```

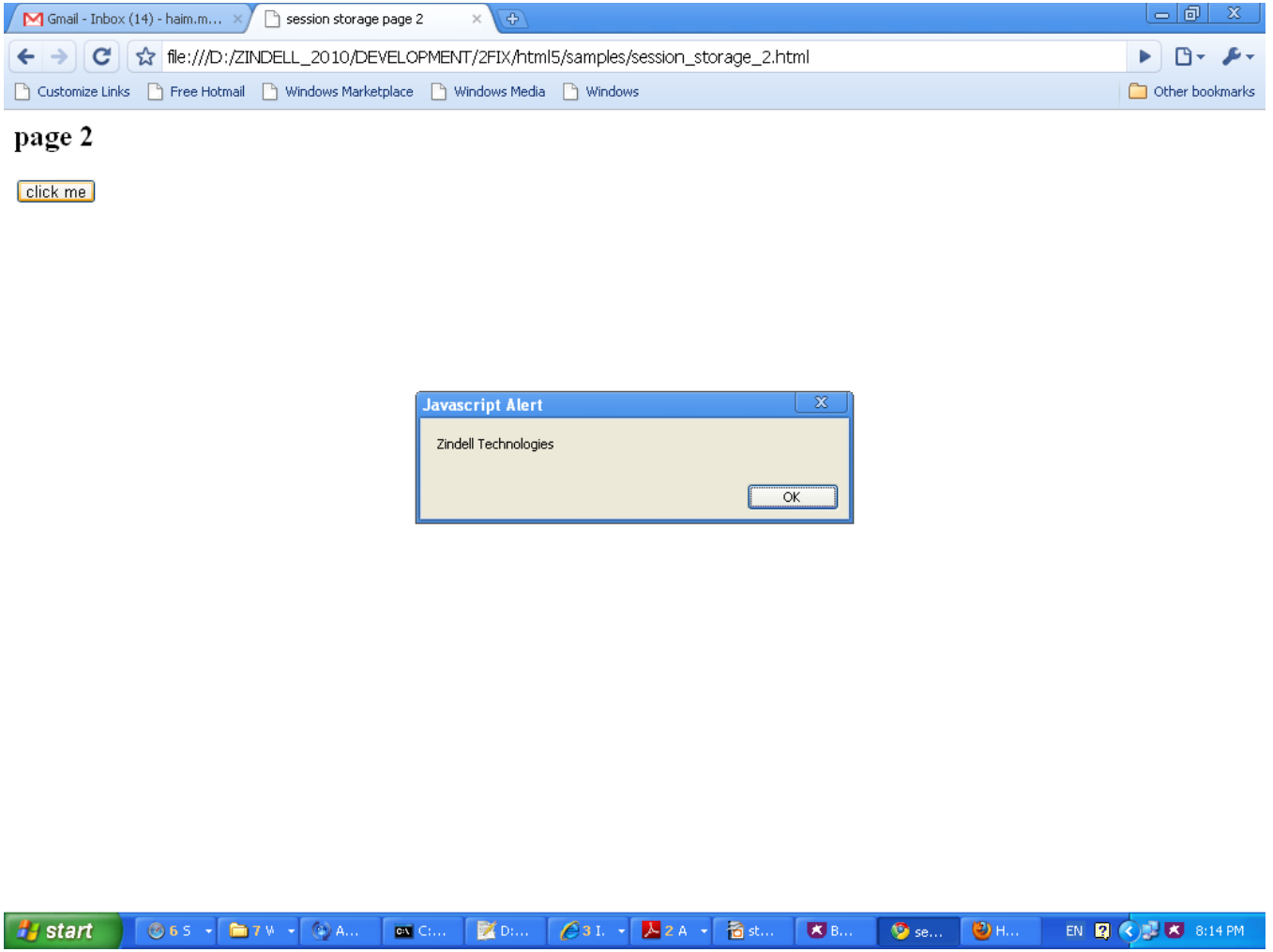
Session Storage

```
<html>
    <head>
        <title>session storage page 2</title>
    </head>
    <body>
        <h2>page 2</h2>
        <script language="javascript">
            function showData()
            {
                alert(sessionStorage.getItem('company'));
            }
        </script>
        <form>
            <input type="button" value="click me" onClick="showData()">
        </form>
    </body>
</html>
```

Session Storage

Session Storage

Session Storage



WebSocket

Introduction

- HTML 5 WebSockets defines a communication channel that operates over the web and allows both direction communication over a single socket.
- Using HTML 5 WebSockets we can dramatically reduce unnecessary network traffic and latency.

Introduction

- Using HTML 5 WebSckets, when data changes on the web server the web server can send a request to the client. We no longer need to implement a client that polls the server.

Web Browser Support

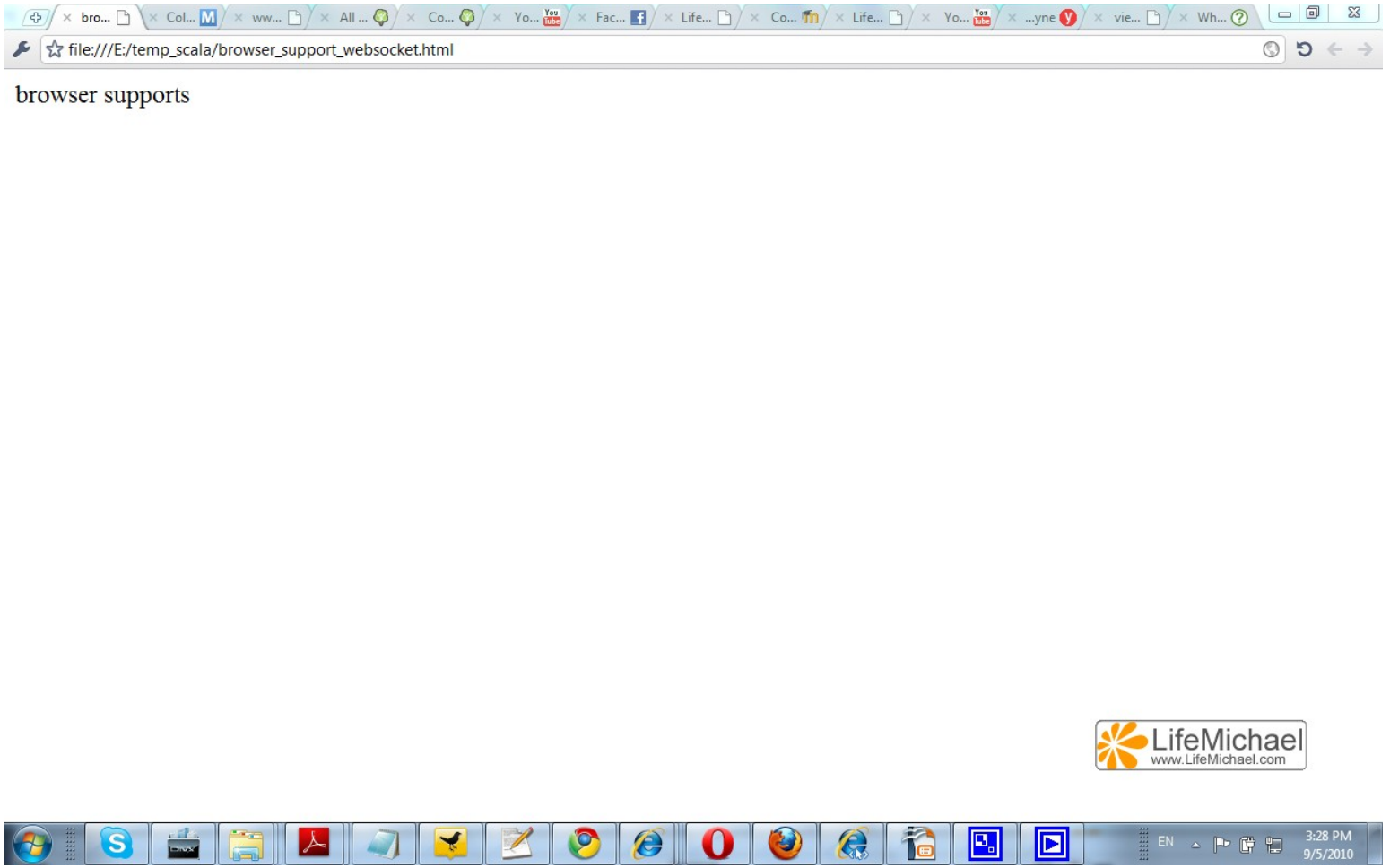
- We can easily check whether the web browser supports WebSocket or not.

```
if (window.WebSocket)
{
    ...
}
```

Web Browser Support

```
<div id="msg"></div>
<script>
if (window.WebSocket)
{
    document.getElementById("msg").innerHTML = "browser supports";
}
else
{
    document.getElementById("msg").innerHTML = "browser doesn't support";
}
</script>
```

Web Browser Support



Creating Web Socket

```
var ws = new WebSocket(“ws://services.abelski.com/samples“);
```

We should pass over to the WebSocket constructore the URL address of the web socket server we intend to use. That address should start with 'ws' which stands for Web Sockets.

Call Back Functions

```
ws.onopen = function(event)
{
    ...
}
```

This function will be called when the connection is established.

Call Back Functions

```
ws.onmessage = function(event)
{
    alert(event.data);
    ...
}
```

This function will be called when a message arrives from the server.

Call Back Functions

```
ws.onclose = function(event)
{
    ...
}
```

This function will be called when the connection is closed.

Sending Data

```
ws.postMessage(“this is the message sent to the server“);
```

This function will be called when the connection is closed.

Close Connection

```
ws.disconnect();
```

Calling this function will disconnect the connection with the server.

Canvas

Overview

- The Canvas element allows us to draw 2D graphics on our web page.
- It is a rectangular area we control each one of its pixels.

```
...
<canvas id="myCanvas" width="200" height="100"></canvas>
...
```

Drawing on Canvas

Once the Canvas was created we can draw various graphics by calling various JavaScript methods on its context.

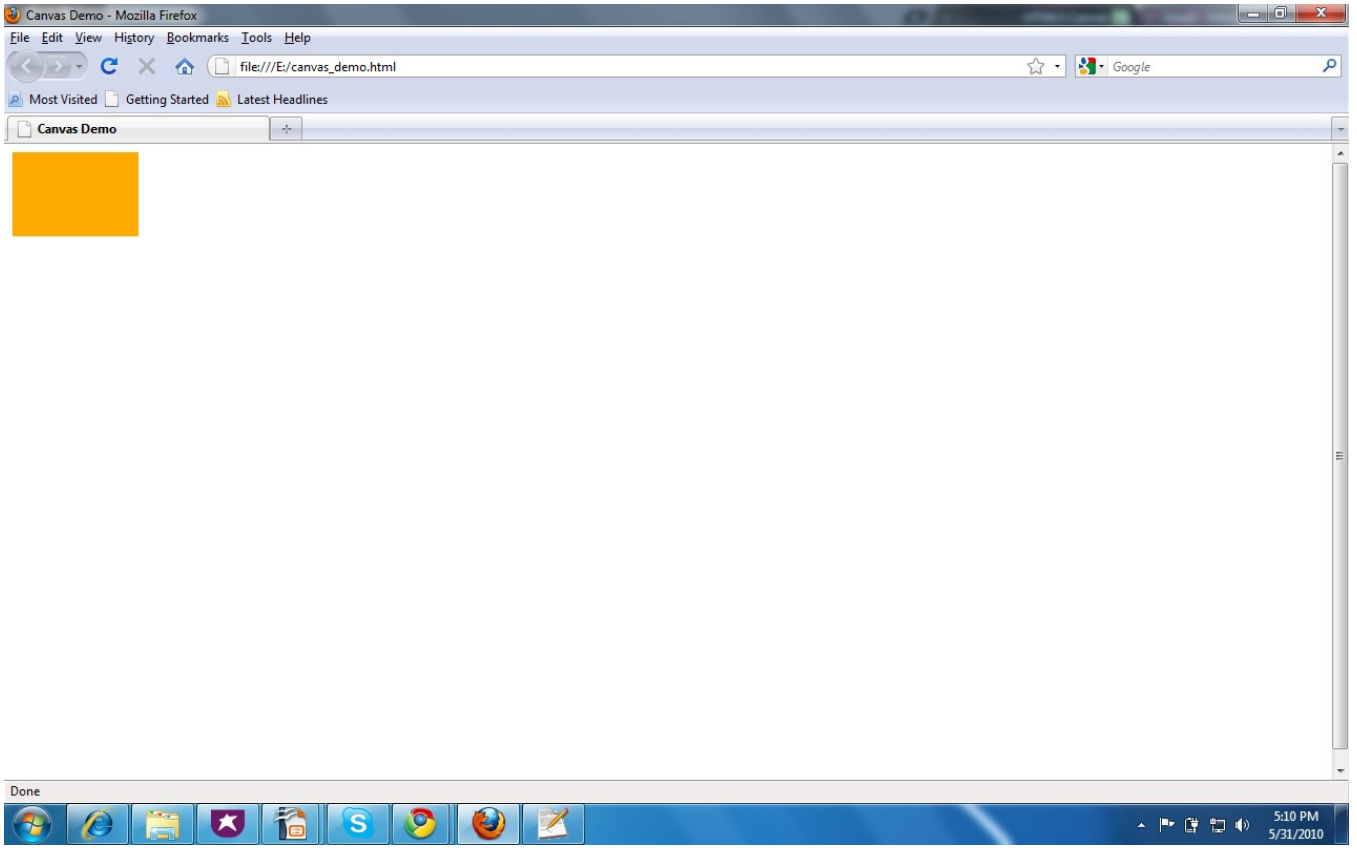
```
...
<canvas id="my_canvas" width="800" height="600">
</canvas>
...
<script type="text/javascript">
    var c=document.getElementById("my_canvas");
    var context=c.getContext("2d");
    context.fillStyle="#FFAA00";
    context.fillRect(0,0,120,80);
</script>

...
```

Drawing on Canvas

```
<html>
    <head>
        <title>Canvas Demo</title>
    </head>
    <body>
        <canvas id="my_canvas" width="800" height="600">
        </canvas>
        <script type="text/javascript">
            var c=document.getElementById("my_canvas");
            var context=c.getContext("2d");
            context.fillStyle="#FFAA00";
            context.fillRect(0,0,120,80);
        </script>
    </body>
</html>
```

Drawing on Canvas

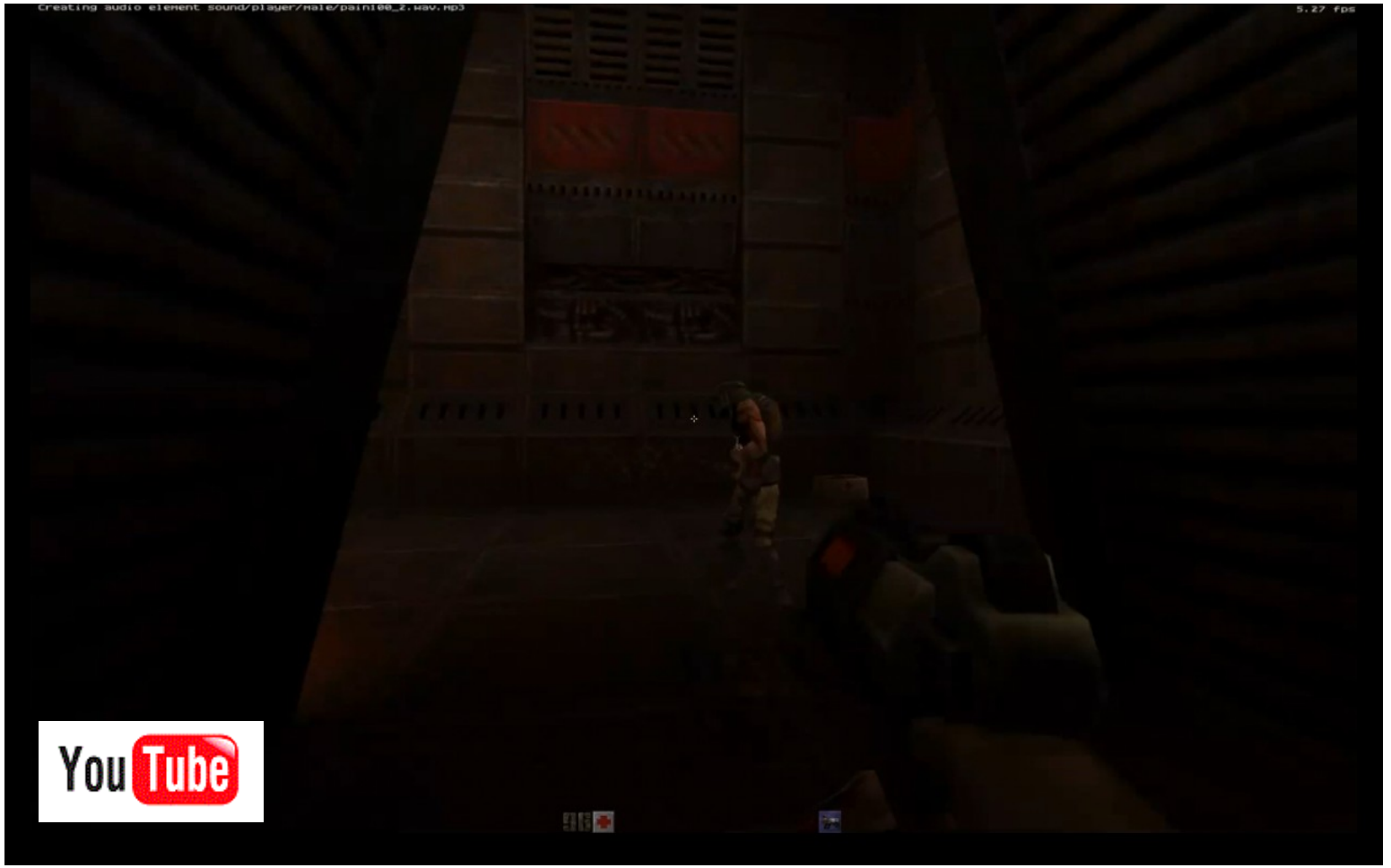


WebGL 3D Graphics

Introduction

- WebGL is an API for 3D graphics within the web browser. OpenGL leads the WebGL standard.
- Similarly to 2D graphics we should call the `getContext` method on our canvas in order to get the context object we can later use for creating the 3D graphics.

Introduction



OpenGL ES 2

- The WebGL implementation is OpenGL ES 2, Khronos royalty-free, cross platform API for full function 2D and 3D graphics on embedded systems.

Foundation Layer

- Similarly to DOM that served as a fundamental layer for the evolve of JavaScript libraries, so is expected with the WebGL.

GLGE

- The GLGE is a javascript library intended to ease the use of WebGL.

  www.glge.org

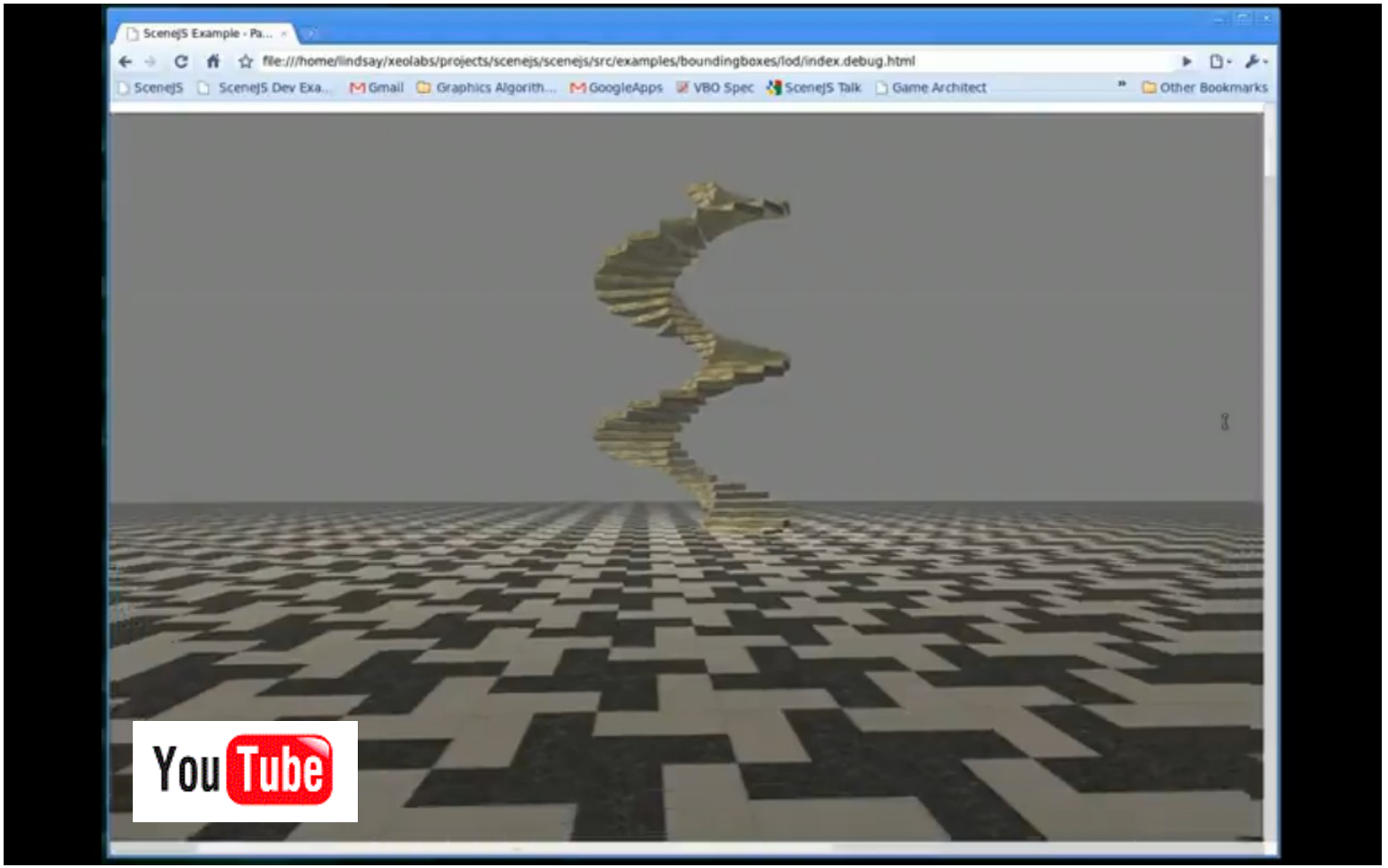
GLGE

SceneJS

- The SceneJS is a javascript library intended to ease the use of WebGL.

  www.scenejs.org

SceneJS



Multimedia

Video

- HTML 5.0 provides a standard for showing video. Using the `<video>` element we can easily embed video within our web page.
- The video formats the `<video>` element supports include the following:

  MPG4 (with H.264 video codec and AAC audio codec)

  OGG (with Thedora video codec and Vorbis audio codec)

Video

Content we place in between the tags will be displayed when the browser doesn't support displaying video

```
<video src="myvid.ogg" controls="controls">

</video>
```

We can use the `width` and `height` attributes in order to specify the size

the `control` attribute is for adding the play, pause and volume controls

Video

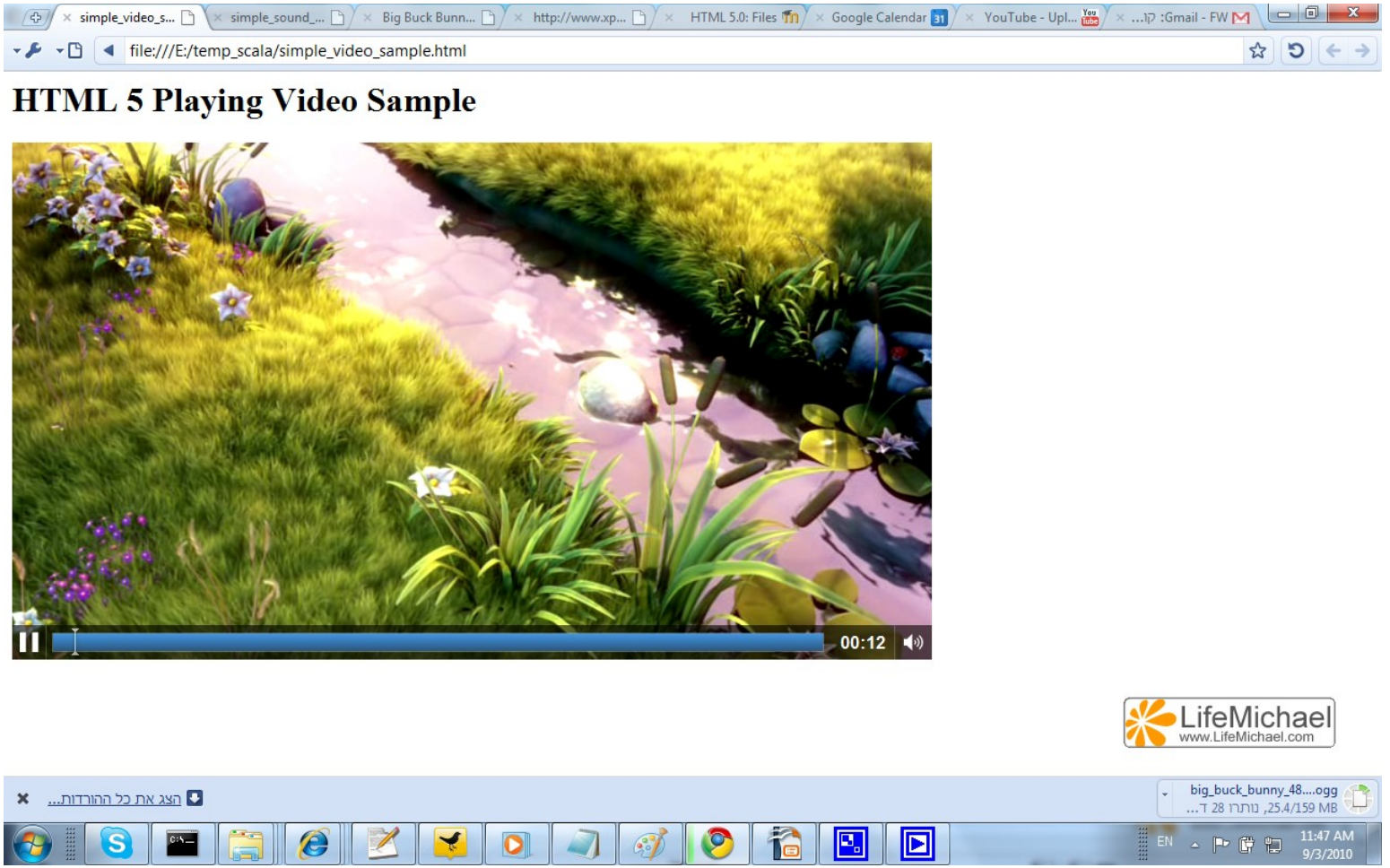
```
<h1>HTML 5 Playing Video Sample</h1>

<video
    src="http://mirror.bigbuckbunny.de/peach/bigbuckbunny_movies/big_buck_bunny_480p_stereo.ogg"
    controls="controls"
    width="854"
    height="480">

    browser does not support html 5.0

</video>
```

Video



Video

- The HTML 5.0 specification supports the following attributes:

`autoplay`

we assign it with the value automatic in order to specify that we want the video to start playing as soon as it is ready.

`controls`

we can assign it with the value controls in order to specify that we want to have the video controls displayed on screen.

`height`

we can specify the height of the rectangle through which the video will be displayed.

Video

`width`

we can specify the width of the rectangle through which the video will be displayed.

`loop`

we can specify the number of times we want the video to be played.

`preload`

we can assign the preload value and by doing so specify that we want the video to be loaded when the page loads.

`src`

We use this attribute in order to specify the exact video file we want to play.

Audio

- The HTML 5.0 specification allows us playing sound using the `<audio>` element.

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

the control attribute adds the play, pause and volume controls

- The `<audio>` element can play sound files or an audio stream.

Audio

- The HTML 5.0 specification aims at supporting the following sounds formats: MP3, WAV and Ogg Vorbis.

Audio

- We can add the `<source>` child elements in between the audio element tags. The browser will use the first supported format.

```
...
<audio controls="controls">
  <source src="mymusic.ogg" type="audio/ogg" />
  <source src="mymusic.mp3" type="audio/mpeg" />
    browser does not support html 5.0
</audio>
...
```

Audio

- The HTML 5.0 specification supports the following audio attributes:

`autoplay`

we assign it with the value autoplay in order to specify that we want the audio to start playing as soon as it is ready.

`controls`

we can assign it with the value controls in order to specify that we want to have the audio controls displayed on screen.

`loop`

we can specify the number of times we want the audio to be played.

Audio

`preload`

we can assign it with preload in order to specify that we want the audio to be loaded together with the page.

`src`

we assign this attribute with the URI of the audio file.

Audio

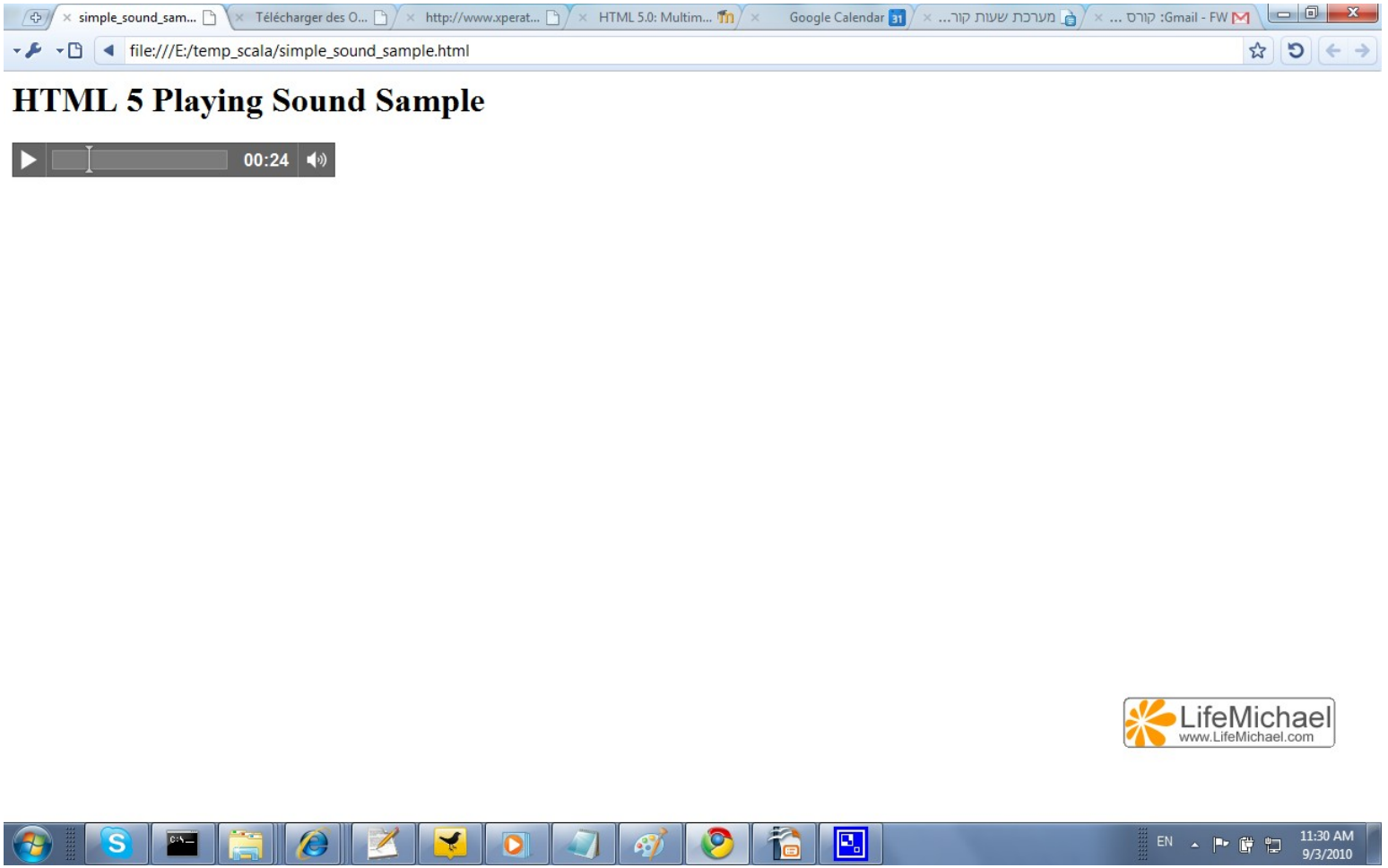
```
<h1>HTML 5 Playing Sound Sample</h1>

<audio controls="controls">
  <source  src="antony_raijekov_views_track_04_jazzabel.ogg"
           type="audio/ogg">

    browser does not support html 5.0

</audio>
```

Audio



Web Workers

Introduction

- The HTML 5 Web Workers provides background processing capabilities. We can use the Web Workers API for running separated threads concurrently with the main scripts in our web page.
- The Web Workers API is especially useful in the prevention of user messages such as the 'unresponsive script' message.

Limitations

- The code executed in a separated thread using the Web Workers API cannot access neither the web page nor its document object model.

Worker

- In order to get a specific JavaScript code executed concurrently in a separated thread we should instantiate the `Worker` type passing over the name of the file that includes the JavaScript code we want to execute in a separated thread.

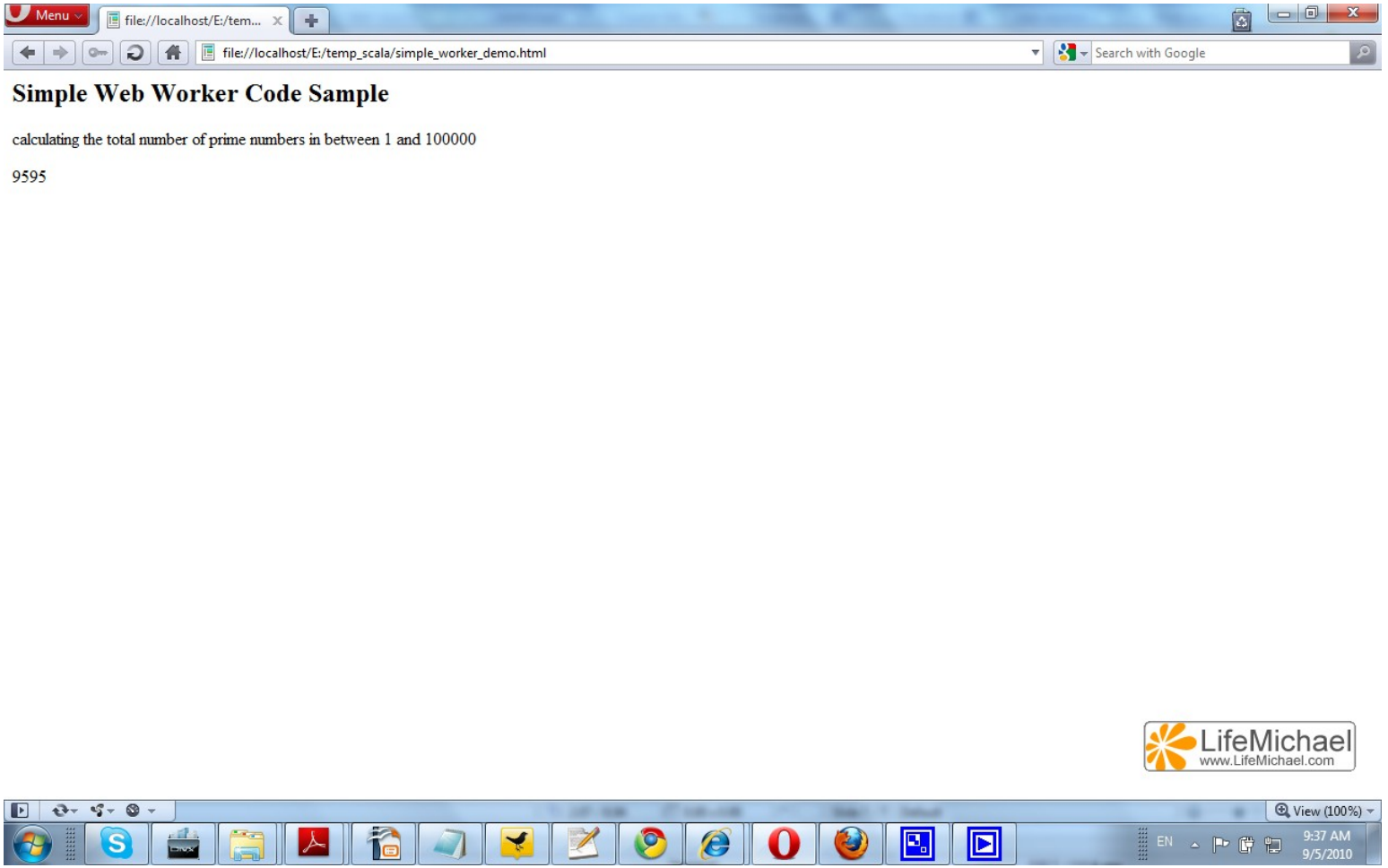
Sample

```
<h2>Simple Web Worker Code Sample</h2>
<p>calculating the total number of prime numbers in between 1 and 100000</p>
<div id="total">1</div>
<script>
    var worker = new Worker('background.js');
    worker.onmessage = updateResult;
    function updateResult(event)
    {
        document.getElementById('total').innerHTML = event.data;
    };
</script>
```

Sample

```
var total = 2;
outer: for(var n=1;n<=100000;n++)
{
    for (var i = 2; i <= Math.sqrt(n); i += 1)
    {
        if (n % i == 0) continue outer;
    }
    total++;
    postMessage(total);
}
```

Sample



Communication

Introduction

- Using the Communication API we can communicate between separated windows, tabs and iframes.

The PostMessage API

- Calling the `postMessage` on the object that represents the other window will send the message to that other window.

The PostMessage API

The second argument specifies the origin the otherWindow must be for the event to be dispatched. It can be either a literal string "*" (which means no preference) or a URI. When the event is dispatched the scheme, the hostname and the port of the otherWindow must match the ones of the otherWindow. Otherwise, the event will not be dispatched.

```
otherWindow.postMessage(message, targetOrigin);
```

This is the string data we want to send over to the other window.

This is the reference to the other window. It can be a reference obtained using the contentWindow property of an iframe element, the object returned by window.open, or the object we get when referring the window.frames array either using a name or a numeric index.

Event Listener

- The other window/tab/iframe should add an event listener for getting the message.

This is the function we want to register.

```
window.addEventListener("message", receiveMessage, false);
```

This is a string representing the specific event type to listen to.

Specifying whether the event handling method will function as a capture meaning that all events of this type will be dispatched to the registered handling method before been dispatched to other handlers beneath the specified handing method.

Event Listener

- The event listener method receives an argument that includes the `data`, the `origin` and the `source` properties.
- The `data` is the received message, the `origin` describes the window/tab/iframe from which the message was received and the `source` refers to the window object that sent the message.

Sample

```
<input type="text" id="msg"/>
<button id="send">Send</button>
<script>
document.getElementById("send").addEventListener("click",send_msg,true);
function send_msg()
{
    var text=document.getElementById("msg").value;
    document.getElementById("inner").
        contentWindow.postMessage(text,"http://www.abelski.com");
}
</script>
<h2>the iframe</h2>
<iframe id="inner" src="inner.html"></iframe>
```

Sample

```
<br>message data:<div id="received_msg"></div>
<br>message origin:<div id="src"></div>
<script>
function handlingMsg(e)
{
    document.getElementById('src').innerHTML = e.origin;
    if(e.origin == "http://www.abelski.com")
    {
        document.getElementById('received_msg').innerHTML = e.data;
    }
}
addEventListener("message",handlingMsg,true);
document.getElementById('received_msg').innerHTML="...";
document.getElementById('src').innerHTML="...";
</script>
```

inner.html

Sample