HTML5

```
<!DOCTYPE HTML>
<html>
<body>

<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
  Your browser does not support the video tag.
</video>

</body>
</html>
```

The HTML5 <!DOCTYPE>

```
<!DOCTYPE html>
```

Minimum HTML5 Document

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Title of the document</title>
</head>

<body>
Content of the document.....
</body>

</html>
```

HTML5 - New Features

- The <canvas> element for 2D drawing
- The <video> and <audio> elements for media playback
- Support for local storage
- New content-specific elements, like <article>, <footer>, <header>, <nav>, <section>
- New form controls, like calendar, date, time, email, url, search

New Elements in HTML5

The New <canvas> Element

<canvas>	Used to draw graphics, on the fly, via scripting (usually JavaScript)

New Media Elements

<audio>	Defines sound content
<video>	Defines a video or movie
<source>	Defines multiple media resources for <video> and <audio>
<embed>	Defines a container for an external application or interactive content (a plug-in)
<track>	Defines text tracks for <video> and <audio>

New Form Elements

<datalist>	Specifies a list of pre-defined options for input controls
<keygen>	Defines a key-pair generator field (for forms)
<output>	Defines the result of a calculation

New Semantic/Structural Elements

<article>	Defines an article
<aside>	Defines content aside from the page content
<bdi>	Isolates a part of text that might be formatted in a different direction from other text outside it
<command>	Defines a command button that a user can invoke
<details>	Defines additional details that the user can view or hide
<dialog>	Defines a dialog box or window
<summary>	Defines a visible heading for a <details> element
<figure>	Specifies self-contained content, like illustrations, diagrams, photos, code listings, etc.
<figcaption>	Defines a caption for a <figure> element
<footer>	Defines a footer for a document or section
<header>	Defines a header for a document or section
<mark>	Defines marked/highlighted text
<meter>	Defines a scalar measurement within a known range (a gauge)
<nav>	Defines navigation links

<progress>	Represents the progress of a task
<ruby>	Defines a ruby annotation (for East Asian typography)
<rt>	Defines an explanation/pronunciation of characters (for East Asian typography)
<rp>	Defines what to show in browsers that do not support ruby annotations
<section>	Defines a section in a document
<time>	Defines a date/time
<wbr>	Defines a possible line-break

HTML5 Canvas

Create a Canvas

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

<canvas id="myCanvas" width="200" height="100"
style="border:1px solid #000000;">
</canvas>
```

Draw Onto The Canvas With JavaScript

```
<script>
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.fillStyle="#FF0000";
ctx.fillRect(0,0,150,75);
</script>
```

FIND the <canvas> element:

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.fillStyle="#FF0000";
ctx.fillRect(0,0,150,75);
```

Canvas - Paths

To draw straight lines on a canvas, we will use the following two methods:

- moveTo(*x*,*y*) defines the starting point of the line
- lineTo(*x*,*y*) defines the ending point of the line

To actually draw the line, we must use one of the "ink" methods, like stroke().

Example

Define a starting point in position (0,0), and an ending point in position (200,100). Then use the stroke() method to actually draw the line:

JavaScript:

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.moveTo(0,0);
ctx.lineTo(200,100);
ctx.stroke();
```

To draw a circle on a canvas, we will use the following method:

- arc(x,y,r,start,stop)

To actually draw the circle, we must use one of the "ink" methods, like stroke() or fill().

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.beginPath();
ctx.arc(95,50,40,0,2*Math.PI);
ctx.stroke();
```

Canvas – Text

To draw text on a canvas, the most important property and methods are:

- font - defines the font properties for text
- fillText(*text,x,y*) - Draws "filled" text on the canvas
- strokeText(*text,x,y*) - Draws text on the canvas (no fill)

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.font="30px Arial";
ctx.fillText("Hello World",10,50);
```

Using strokeText():

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.font="30px Arial";
ctx.strokeText("Hello World",10,50);
```

Canvas - Gradients

There are two different types of gradients:

- createLinearGradient(*x*,*y*,*x1*,*y1*) - Creates a linear gradient
- createRadialGradient(*x*,*y*,*r*,*x1*,*y1*,*r1*) - Creates a radial/circular gradient
- The addColorStop() method specifies the color stops, and its position along the gradient. Gradient positions can be anywhere between 0 to 1.

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");

// Create gradient
var grd=ctx.createLinearGradient(0,0,200,0);
grd.addColorStop(0,"red");
grd.addColorStop(1,"white");

// Fill with gradient
ctx.fillStyle=grd;
ctx.fillRect(10,10,150,80);
```

Using createRadialGradient():

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");

// Create gradient
var grd=ctx.createRadialGradient(75,50,5,90,60,100);
grd.addColorStop(0,"red");
grd.addColorStop(1,"white");

// Fill with gradient
ctx.fillStyle=grd;
ctx.fillRect(10,10,150,80);
```

Canvas - Images

To draw an image on a canvas, we will use the following method:

- drawImage(*image,x,y*)

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
var img=document.getElementById("scream");
ctx.drawImage(img,10,10);
```

HTML5 Inline SVG

What is SVG?

- SVG stands for Scalable Vector Graphics
- SVG is used to define vector-based graphics for the Web
- SVG defines the graphics in XML format
- SVG graphics do NOT lose any quality if they are zoomed or resized
- Every element and every attribute in SVG files can be animated
- SVG is a W3C recommendation

SVG Advantages

Advantages of using SVG over other image formats (like JPEG and GIF) are:

- SVG images can be created and edited with any text editor
- SVG images can be searched, indexed, scripted, and compressed
- SVG images are scalable
- SVG images can be printed with high quality at any resolution
- SVG images are zoomable (and the image can be zoomed without degradation)

Embed SVG Directly Into HTML Pages

```
<!DOCTYPE html>
<html>
<body>

<svg xmlns="http://www.w3.org/2000/svg" version="1.1" height="190">
  <polygon points="100,10 40,180 190,60 10,60 160,180"
  style="fill:lime;stroke:purple;stroke-width:5;fill-rule:evenodd;">
</svg>
```

```
</body>
</html>
```

Differences Between SVG and Canvas

SVG is a language for describing 2D graphics in XML.

Canvas draws 2D graphics, on the fly (with a JavaScript).

SVG is XML based, which means that every element is available within the SVG DOM. You can attach JavaScript event handlers for an element.

In SVG, each drawn shape is remembered as an object. If attributes of an SVG object are changed, the browser can automatically re-render the shape.

Canvas is rendered pixel by pixel. In canvas, once the graphic is drawn, it is forgotten by the browser. If its position should be changed, the entire scene needs to be redrawn, including any objects that might have been covered by the graphic.

Canvas	SVG
• Resolution dependent • No support for event handlers • Poor text rendering capabilities • You can save the resulting image as .png or .jpg • Well suited for graphic-intensive games	• Resolution independent • Support for event handlers • Best suited for applications with large rendering areas (Google Maps) • Slow rendering if complex (anything that uses the DOM a lot will be slow) • Not suited for game applications

HTML5 Drag and Drop

Drag and drop is a part of the HTML5 standard.

HTML5 Drag and Drop Example

```
<!DOCTYPE HTML>
<html>
<head>
<script>
function allowDrop(ev)
{
ev.preventDefault();
```

```
}

function drag(ev)
{
ev.dataTransfer.setData("Text",ev.target.id);
}

function drop(ev)
{
ev.preventDefault();
var data=ev.dataTransfer.getData("Text");
ev.target.appendChild(document.getElementById(data));
}
</script>
</head>
<body>

<div id="div1" ondrop="drop(event)"
ondragover="allowDrop(event)"></div>

<img id="drag1" src="img_logo.gif" draggable="true"
ondragstart="drag(event)" width="336" height="69">

</body>
</html>
```

Make an Element Draggable

```
<img draggable="true">
```

What to Drag - ondragstart and setData()

Then, specify what should happen when the element is dragged.

In the example above, the ondragstart attribute calls a function, drag(event), that specifies what data to be dragged.

The dataTransfer.setData() method sets the data type and the value of the dragged data:

```
function drag(ev)
{
ev.dataTransfer.setData("Text",ev.target.id);
}
```

Where to Drop - ondragover

The ondragover event specifies where the dragged data can be dropped.

By default, data/elements cannot be dropped in other elements. To allow a drop, we must prevent the default handling of the element.

This is done by calling the event.preventDefault() method for the ondragover event:

```
event.preventDefault()
```

Do the Drop - ondrop

When the dragged data is dropped, a drop event occurs.

In the example above, the ondrop attribute calls a function, drop(event):

```
function drop(ev)
{
ev.preventDefault();
var data=ev.dataTransfer.getData("Text");
ev.target.appendChild(document.getElementById(data));
}
```

Code explained:

- Call preventDefault() to prevent the browser default handling of the data (default is open as link on drop)
- Get the dragged data with the dataTransfer.getData("Text") method. This method will return any data that was set to the same type in the setData() method
- The dragged data is the id of the dragged element ("drag1")
- Append the dragged element into the drop element

HTML5 Geolocation

HTML5 - Using Geolocation

Use the getCurrentPosition() method to get the user's position.

The example below is a simple Geolocation example returning the latitude and longitude of the user's position:

```
<script>
var x=document.getElementById("demo");
function getLocation()
 {
 if (navigator.geolocation)
   {
   navigator.geolocation.getCurrentPosition(showPosition);
   }
 else{x.innerHTML="Geolocation is not supported by this browser.";}
 }
function showPosition(position)
 {
 x.innerHTML="Latitude: " + position.coords.latitude +
 "<br>Longitude: " + position.coords.longitude;
 }
</script>
```

Example explained:

- Check if Geolocation is supported
- If supported, run the getCurrentPosition() method. If not, display a message to the user
- If the getCurrentPosition() method is successful, it returns a coordinates object to the function specified in the parameter (showPosition)
- The showPosition() function gets the displays the Latitude and Longitude

The example above is a very basic Geolocation script, with no error handling.

Handling Errors and Rejections

```
function showError(error)
 {
 switch(error.code)
   {
   case error.PERMISSION_DENIED:
     x.innerHTML="User denied the request for Geolocation."
     break;
   case error.POSITION_UNAVAILABLE:
     x.innerHTML="Location information is unavailable."
     break;
   case error.TIMEOUT:
     x.innerHTML="The request to get user location timed out."
     break;
   case error.UNKNOWN_ERROR:
     x.innerHTML="An unknown error occurred."
     break;
```

```
  }
}
```

Displaying the Result in a Map

To display the result in a map, you need access to a map service that can use latitude and longitude, like Google Maps:

Example

```
function showPosition(position)
{
var latlon=position.coords.latitude+","+position.coords.longitude;

var img_url="http://maps.googleapis.com/maps/api/staticmap?center="
+latlon+"&zoom=14&size=400x300&sensor=false";

document.getElementById("mapholder").innerHTML="<img src='"+img_url+"'>";
}
```

Location-specific Information

This page demonstrated how to show a user's position on a map. However, Geolocation is also very useful for location-specific information.

Examples:

- Up-to-date local information
- Showing Points-of-interest near the user
- Turn-by-turn navigation (GPS)

The getCurrentPosition() Method - Return Data

The getCurrentPosition() method returns an object if it is successful. The latitude, longitude and accuracy properties are always returned. The other properties below are returned if available.

Property	Description
coords.latitude	The latitude as a decimal number
coords.longitude	The longitude as a decimal number
coords.accuracy	The accuracy of position

coords.altitude	The altitude in meters above the mean sea level
coords.altitudeAccuracy	The altitude accuracy of position
coords.heading	The heading as degrees clockwise from North
coords.speed	The speed in meters per second
timestamp	The date/time of the response

Geolocation object - Other interesting Methods

watchPosition() - Returns the current position of the user and continues to return updated position as the user moves (like the GPS in a car).

clearWatch() - Stops the watchPosition() method.

The example below shows the watchPosition() method. You need an accurate GPS device to test this (like iPhone):

Example

```
<script>
var x=document.getElementById("demo");
function getLocation()
  {
  if (navigator.geolocation)
    {
    navigator.geolocation.watchPosition(showPosition);
    }
  else{x.innerHTML="Geolocation is not supported by this browser.";}
  }
function showPosition(position)
  {
  x.innerHTML="Latitude: " + position.coords.latitude +
  "<br>Longitude: " + position.coords.longitude;
  }
</script>
```

HTML5 Video

HTML5 Video - How It Works

To show a video in HTML5, this is all you need:

Example

```
<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
Your browser does not support the video tag.
</video>
```

MIME Types for Video Formats

Format	MIME-type
MP4	video/mp4
WebM	video/webm
Ogg	video/ogg