HTML5

HTML5 is the latest version of HTML.

HTML5

1. HTML5 New Elements
2. HTML5 Removed Elements
3. HTML5 Drag/Drop
4. HTML5 Geolocation
5. HTML5 Video
6. HTML5 Audio
7. HTML5 Input Types
8. HTML5 Form Elements
9. HTML5 Form Attributes
10. HTML5 Web Storage
11. HTML5 App Cache
12. HTML5 Web Workers
13. HTML5 SSE
14. HTML5 Canvas
15. HTML5 SVG

---

1) Canvas:

- It is a graphics container, in which you can draw 2D graphics.
- To draw graphics like shapes, lines, ellipses etc.
- We create a canvas element using HTML 5
- We use pure java script to draw graphics on that HTML 5 canvas.

2) SVG:

- Scalable Vector Graphics
- It is an another way of drawing graphics.
- SVG can be programmed using XML.
- Here also, target is to draw graphics

3) HTML5 Drag/Drop:

- We can drag and drop an element into another element such as div.
- We can allow user to drag and drop images into a box.
- For example, the user can drag and drop an img into div.

4) HTML5 Geolocation:

- We can track the current location of the client, where our page is running.
- Works good in mobiles

5) HTML5 Video

- Basically, we are currently depending on 3rd party video players such as flash, yahoo player etc., to play videos
- We show video players directly using html 5

6) HTML5 Audio

- Basically, we are currently depending on 3rd party audio players such as flash etc.

- We show audio players directly using html 5

7) HTML5 Input Types:

- <input type="date" />
- <input type="email" />
- <input type="range" />
- <input type="color" />
- etc.

9) HTML5 Form Attributes:

- These are the new attributes are introduced in <form> and <input> tags.
- Example 1:
  - ❖ <input type=”submit” **formaction**=”serverpage1.asp” />
  - ❖ <input type=”submit” **formaction**=”serverpage2.asp” />
- Example 2:
  - ❖ <input type="file" **multiple="multiple"**>

10) HTML5 Web Storage

- We can store some data temporarily, while the page is running.
- **Data:** simple values / objects / local databases
- It is used to store temporary data only.
- We can programmatically add data / erase data.
- The data stored using one page, can be accessible using that page only.
- This is like an enhancement for cookies.

11) HTML5 App Cache:

- It is also known as offline applications.

- Works like browser cache.
- If you enable this feature for a page, even though net is disconnected, that page can be loaded on the browser.
- It internally stores the page output in the cache memory.

HTML5 Web Workers:

- This is used to do background work, while the page is running, without disturbing the functionality of the page.
- Examples: Count down, Time updates etc.

HTML5 SSE:

- Server Side Events
- Server itself can send some notification / data to the browser, while the page is running on the browser.

Drawback of HTML 5:

- Browser incompatibility
- Still it is under construction
- All major browsers like IE9, Mozilla Firefox, Google Chrome, Opera, Safari etc., have **partial support** for HTML 5.

HTML5 is not yet an official standard, and no browsers have full HTML5 support.

All major browsers (Safari, Chrome, Firefox, Opera, Internet Explorer) continue to add new HTML5 features to their latest versions.

What is HTML5?

- HTML5 will be the new standard for HTML.
- The previous version of HTML, HTML 4.01, came in 1999. The web has changed a lot since then.
- HTML5 is still a work in progress. However, the major browsers support many of the new HTML5 elements and APIs.

How Did HTML5 Get Started?

- HTML5 is a co-operation between the World Wide Web Consortium (W3C) and the Web Hypertext Application Technology Working Group (WHATWG).
- WHATWG was working with web forms and applications, and W3C was working with XHTML 2.0. In 2006, they decided to cooperate and create a new version of HTML.

Some rules for HTML5 were established:

- New features should be based on HTML, CSS, DOM, and JavaScript
- Reduce the need for external plug-ins (like Flash)
- Better error handling
- More markup to replace scripting
- HTML5 should be device independent
- The development process should be visible to the public

The HTML5 <!DOCTYPE>

In HTML5 there is only one <!doctype> declaration, and it is very simple:

<!DOCTYPE html>

Minimum HTML5 Document

Below is a simple HTML5 document, with the minimum of required tags:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Title of the document</title>
    </head>

   <body>
        The content of the document.....
   </body>

</html>
```

New Elements in HTML5

The internet has changed a lot since HTML 4.01 became a standard in 1999.

Today, some elements in HTML 4.01 are obsolete, never used, or not used the way they were intended to. These elements are removed or re-written in HTML5.

To better handle today's internet use, HTML5 includes new elements for better structure, better form handling, drawing, and for media content.

New Semantic/Structural Elements

HTML5 offers new elements for better structure:

Tag	Description
<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
<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
<hgroup>	Groups a set of <h1> to <h6> elements when a heading has multiple levels
<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

New Media Elements

HTML5 offers new elements for media content:

Tag	Description
<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>

The new <canvas> Element

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

New Form Elements

HTML5 offers new form elements, for more functionality:

Tag	Description
<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

Removed Elements

The following HTML 4.01 elements are removed from HTML5:

- **<acronym>**
- **<applet>**
- **<basefont>**
- **<big>**
- ** **
- **<dir>**
- **<font>**
- **<frame>**
- **<frameset>**
- **<noframes>**
- **<strike>**
- **<tt>**

HTML5 Drag and Drop

Drag and drop is a very common feature. It is when you "grab" an object and drag it to a different location.

In HTML5, drag and drop is part of the standard, and any element can be draggable.

Browser Support

Internet Explorer 9, Firefox, Opera 12, Chrome, and Safari 5 support drag and drop.

Note: Drag and drop does not work in Safari 5.1.2.

HTML5 Drag and Drop Example

The example below is a simple drag and drop example:

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>
```

It might seem complicated, but let's go through all the different parts of a drag and drop event.

Make an Element Draggable

First of all: To make an element draggable, set the draggable attribute to true:

```
<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);
}
```

In this case, the data type is "Text" and the value is the id of the draggable element ("drag1").

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 Geolocation is used to locate a user's position

Locate the User's Position

The HTML5 Geolocation API is used to get the geographical position of a user.

Since this can compromise user privacy, the position is not available unless the user approves it.

Browser Support

Internet Explorer 9, Firefox, Chrome, Safari and Opera support Geolocation.

Note: Geolocation is much more accurate for devices with GPS, like iPhone.

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:

Example

```
<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

The second parameter of the getCurrentPosition() method is used to handle errors. It specifies a function to run if it fails to get the user's location:

Example

```
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;
  }
 }
```

Error Codes:

- Permission denied - The user did not allow Geolocation
- Position unavailable - It is not possible to get the current location
- Timeout - The operation timed out

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+"'>";
}
```

In the example above we use the returned latitude and longitude data to show the location in a Google map (using a static image).

Google Map Script
How to use a script to show an interactive map with a marker, zoom and drag options.

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

Many modern websites show videos. HTML5 provides a standard for showing them.

Video on the Web

Until now, there has not been a standard for showing a video/movie on a web page.

Today, most videos are shown through a plug-in (like flash). However, different browsers may have different plug-ins.

HTML5 defines a new element which specifies a standard way to embed a video/movie on a web page: the <video> element.

Browser Support

Internet Explorer 9, Firefox, Opera, Chrome, and Safari support the <video> element.

Note: Internet Explorer 8 and earlier versions, do not support the <video> element.

HTML5 Video - How It Works

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

Example

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

The control attribute adds video controls, like play, pause, and volume.

It is also a good idea to always include width and height attributes. If height and width are set, the space required for the video is reserved when the page is loaded. However, without these attributes, the browser does not know the size of the video, and cannot reserve the appropriate space to it. The effect will be that the page layout will change during loading (while the video loads).

You should also insert text content between the <video> and </video> tags for browsers that do not support the <video> element.

The <video> element allows multiple <source> elements. <source> elements can link to different video files. The browser will use the first recognized format.

Video Formats and Browser Support

Currently, there are 3 supported video formats for the <video> element: MP4, WebM, and Ogg:

Browser	MP4	WebM	Ogg
Internet Explorer 9	YES	NO	NO
Firefox 4.0	NO	YES	YES
Google Chrome 6	YES	YES	YES
Apple Safari 5	YES	NO	NO
Opera 10.6	NO	YES	YES

- MP4 = MPEG 4 files with H264 video codec and AAC audio codec
- WebM = WebM files with VP8 video codec and Vorbis audio codec
- Ogg = Ogg files with Theora video codec and Vorbis audio codec

---

HTML5 <video> - DOM Methods and Properties

HTML5 has DOM methods, properties, and events for the <video> and <audio> elements.

These methods, properties, and events allow you to manipulate <video> and <audio> elements using JavaScript.

There are methods for playing, pausing, and loading, for example and there are properties (like duration and volume). There are also DOM events that can notify you when the <video> element begins to play, is paused, is ended, etc.

The example below illustrate, in a simple way, how to address a <video> element, read and set properties, and call methods.

HTML5 Video Tags

Tag	Description
<video>	Defines a video or movie
<source>	Defines multiple media resources for media elements, such as <video> and <audio>
<track>	Defines text tracks in mediaplayers

HTML5 Audio

HTML5 provides a standard for playing audio files.

Audio on the Web

Until now, there has not been a standard for playing audio files on a web page.

Today, most audio files are played through a plug-in (like flash). However, different browsers may have different plug-ins.

HTML5 defines a new element which specifies a standard way to embed an audio file on a web page: the <audio> element.

Browser Support

Internet Explorer 9, Firefox, Opera, Chrome, and Safari support the <audio> element.

Note: Internet Explorer 8 and earlier versions, do not support the <audio> element.

HTML5 Audio - How It Works

To play an audio file in HTML5, this is all you need:

Example

```
<audio controls="controls">
  <source src="horse.ogg" type="audio/ogg">
  <source src="horse.mp3" type="audio/mpeg">
  Your browser does not support the audio element.
</audio>
```

The control attribute adds audio controls, like play, pause, and volume.

You should also insert text content between the <audio> and </audio> tags for browsers that do not support the <audio> element.

The <audio> element allows multiple <source> elements. <source> elements can link to different audio files. The browser will use the first recognized format.

Audio Formats and Browser Support

Currently, there are 3 supported file formats for the <audio> element: MP3, Wav, and Ogg:

Browser	MP3	Wav	Ogg
Internet Explorer 9	YES	NO	NO
Firefox 4.0	NO	YES	YES
Google Chrome 6	YES	YES	YES
Apple Safari 5	YES	YES	NO
Opera 10.6	NO	YES	YES

HTML5 Audio Tags

Tag	Description
<audio>	Defines sound content
<source>	Defines multiple media resources for media elements, such as <video> and <audio>

HTML5 New Input Types

HTML5 has several new input types for forms. These new features allow better input control and validation.

This chapter covers the new input types:

- color
- date
- datetime
- datetime-local
- email
- month
- number
- range
- search
- tel
- time
- url
- week

Note: Not all major browsers support all the new input types. However, you can already start using them; If they are not supported, they will behave as regular text fields.

Input Type: color

The color type is used for input fields that should contain a color.

Example

Select a color from a color picker:

Select your favorite color: <input type="color" name="favcolor">

Input Type: date

The date type allows the user to select a date.

Example

Define a date control:

Birthday: <input type="date" name="bday">

Input Type: datetime

The datetime type allows the user to select a date and time (with time zone).

Example

Define a date and time control (with time zone):

Birthday (date and time): <input type="datetime" name="bdaytime">

Input Type: datetime-local

The datetime-local type allows the user to select a date and time (no time zone).

Example

Define a date and time control (no time zone):

Birthday (date and time): <input type="datetime-local" name="bdaytime">

Input Type: email

The email type is used for input fields that should contain an e-mail address.

Example

Define a field for an e-mail address (will be automatically validated when submitted):

E-mail: <input type="email" name="usremail">

Tip: Safari on iPhone recognizes the email type, and changes the on-screen keyboard to match it (adds @ and .com options).

Input Type: month

The month type allows the user to select a month and year.

Example

Define a month and year control (no time zone):

Birthday (month and year): <input type="month" name="bdaymonth">

Input Type: number

The number type is used for input fields that should contain a numeric value.

You can also set restrictions on what numbers are accepted:

Example

Define a numeric field (with restrictions):

Quantity (between 1 and 5): <input type="number" name="quantity" min="1" max="5">

Use the following attributes to specify restrictions:

- max - specifies the maximum value allowed
- min - specifies the minimum value allowed
- step - specifies the legal number intervals

- value - Specifies the default value

Try an example with all the restriction attributes: Try it yourself

---

Input Type: range

The range type is used for input fields that should contain a value from a range of numbers.

You can also set restrictions on what numbers are accepted.

Example

Define a control for entering a number whose exact value is not important (like a slider control):

<input type="range" name="points" min="1" max="10">

Use the following attributes to specify restrictions:

- max - specifies the maximum value allowed
- min - specifies the minimum value allowed
- step - specifies the legal number intervals
- value - Specifies the default value

Input Type: search

The search type is used for search fields (a search field behaves like a regular text field).

Example

Define a search field (like a site search, or Google search):

Search Google: <input type="search" name="googlesearch">

Input Type: tel

Example

Define a field for entering a telephone number:

Telephone: <input type="tel" name="usrtel">

Input Type: time

The time type allows the user to select a time.

Example

Define a control for entering a time (no time zone):

Select a time: <input type="time" name="usr_time">

Input Type: url

The url type is used for input fields that should contain a URL address.

The value of the url field is automatically validated when the form is submitted.

Example

Define a field for entering a URL:

Add your homepage: <input type="url" name="homepage">

Tip: Safari on iPhone recognizes the url input type, and changes the on-screen keyboard to match it (adds .com option).

Input Type: week

The week type allows the user to select a week and year.

Example

Define a week and year control (no time zone):

Select a week: <input type="week" name="week_year">

HTML5 New Form Elements

HTML5 has the following new form elements:

- <datalist>
- <keygen>
- <output>

Note: Not all major browsers support all the new form elements. However, you can already start using them; If they are not supported, they will behave as regular text fields.

HTML5 <datalist> Element

The <datalist> element specifies a list of pre-defined options for an <input> element.

The <datalist> element is used to provide an "autocomplete" feature on <input> elements. Users will see a drop-down list of pre-defined options as they input data.

Use the <input> element's list attribute to bind it together with a <datalist> element.

Example

An <input> element with pre-defined values in a <datalist>:

```
<input list="browsers">

<datalist id="browsers">
  <option value="Internet Explorer">
  <option value="Firefox">
  <option value="Chrome">
  <option value="Opera">
```

```
  <option value="Safari">
</datalist>
```

HTML5 <keygen> Element

The purpose of the <keygen> element is to provide a secure way to authenticate users.

The <keygen> tag specifies a key-pair generator field in a form.

When the form is submitted, two keys are generated, one private and one public.

The private key is stored locally, and the public key is sent to the server. The public key could be used to generate a client certificate to authenticate the user in the future.

Example

A form with a keygen field:

```
<form action="demo_keygen.asp" method="get">
Username: <input type="text" name="usr_name">
Encryption: <keygen name="security">
<input type="submit">
</form>
```

HTML5 <output> Element

The <output> element represents the result of a calculation (like one performed by a script).

Example

Perform a calculation and show the result in an <output> element:

```
<form oninput="x.value=parseInt(a.value)+parseInt(b.value)">0
<input type="range" name="a" value="50">100  +
<input type="number" name="b" value="50">=
<output name="x" for="a b"></output>
</form>
```

HTML5 New Form Attributes

HTML5 has several new attributes for <form> and <input>.

New attributes for <form>:

- autocomplete
- novalidate

New attributes for <input>:

- autocomplete
- autofocus
- form
- formaction
- formenctype
- formmethod
- formnovalidate
- formtarget
- height and width
- list

- min and max
- multiple
- pattern (regexp)
- placeholder
- required
- step

<form> / <input> autocomplete Attribute

The autocomplete attribute specifies whether a form or input field should have autocomplete on or off.

When autocomplete is on, the browser automatically complete values based on values that the user has entered before.

Tip: It is possible to have autocomplete "on" for the form, and "off" for specific input fields, or vice versa.

Note: The autocomplete attribute works with <form> and the following <input> types: text, search, url, tel, email, password, datepickers, range, and color.

Example

An HTML form with autocomplete on (and off for one input field):

```
<form action="demo_form.asp" autocomplete="on">
  First name:<input type="text" name="fname"><br>
  Last name: <input type="text" name="lname"><br>
  E-mail: <input type="email" name="email" autocomplete="off"><br>
  <input type="submit">
</form>
```

Tip: In some browsers you may need to activate the autocomplete function for this to work.

<form> novalidate Attribute

The novalidate attribute is a boolean attribute.

When present, it specifies that the form-data (input) should not be validated when submitted.

Example

Indicates that the form is not to be validated on submit:

```
<form action="demo_form.asp" novalidate="novalidate">
  E-mail: <input type="email" name="user_email">
  <input type="submit">
</form>
```

<input> autofocus Attribute

The autofocus attribute is a boolean attribute.

When present, it specifies that an <input> element should automatically get focus when the page loads.

Example

Let the "First name" input field automatically get focus when the page loads:

First name:<input type="text" name="fname" autofocus="autofocus">

<input> form Attribute

The form attribute specifies one or more forms an <input> element belongs to.

Tip: To refer to more than one form, use a space-separated list of form ids.

Example

An input field located outside the HTML form (but still a part of the form):

```
<form action="demo_form.asp" id="form1">
  First name: <input type="text" name="fname"><br>
  <input type="submit" value="Submit">
</form>

Last name: <input type="text" name="lname" form="form1">
```

<input> formaction Attribute

The formaction attribute specifies the URL of a file that will process the input control when the form is submitted.

The formaction attribute overrides the action attribute of the <form> element.

Note: The formaction attribute is used with type="submit" and type="image".

Example

An HTML form with two submit buttons, with different actions:

```
<form action="demo_form.asp">
  First name: <input type="text" name="fname"><br>
  Last name: <input type="text" name="lname"><br>
  <input type="submit" value="Submit"><br>
  <input type="submit" formaction="demo_admin.asp"
  value="Submit as admin">
</form>
```

<input> formenctype Attribute

The formenctype attribute specifies how the form-data should be encoded when submitting it to the server (only for forms with method="post")

The formenctype attribute overrides the enctype attribute of the <form> element.

Note: The formenctype attribute is used with type="submit" and type="image".

Example

Send form-data that is default encoded (the first submit button), and encoded as "multipart/form-data" (the second submit button):

```
<form action="demo_post_enctype.asp" method="post">
  First name: <input type="text" name="fname"><br>
  <input type="submit" value="Submit">
  <input type="submit" formenctype="multipart/form-data"
```

```
 value="Submit as Multipart/form-data">
</form>
```

<input> formmethod Attribute

The formmethod attribute defines the HTTP method for sending form-data to the action URL.

The formmethod attribute overrides the method attribute of the <form> element.

Note: The formmethod attribute can be used with type="submit" and type="image".

Example

The second submit button overrides the HTTP method of the form:

```
<form action="demo_form.asp" method="get">
 First name: <input type="text" name="fname"><br>
 Last name: <input type="text" name="lname"><br>
 <input type="submit" value="Submit">
 <input type="submit" formmethod="post" formaction="demo_post.asp"
 value="Submit using POST">
</form>
```

<input> formnovalidate Attribute

The novalidate attribute is a boolean attribute.

When present, it specifies that the <input> element should not be validated when submitted.

The formnovalidate attribute overrides the novalidate attribute of the <form> element.

Note: The formnovalidate attribute can be used with type="submit".

Example

A form with two submit buttons (with and without validation):

```
<form action="demo_form.asp">
  E-mail: <input type="email" name="userid"><br>
  <input type="submit" value="Submit"><br>
  <input type="submit" formnovalidate="formnovalidate"
  value="Submit without validation">
</form>
```

<input> formtarget Attribute

The formtarget attribute specifies a name or a keyword that indicates where to display the response that is received after submitting the form.

The formtarget attribute overrides the target attribute of the <form> element.

Note: The formtarget attribute can be used with type="submit" and type="image".

Example

A form with two submit buttons, with different target windows:

```
<form action="demo_form.asp">
  First name: <input type="text" name="fname"><br>
  Last name: <input type="text" name="lname"><br>
  <input type="submit" value="Submit as normal">
  <input type="submit" formtarget="_blank"
  value="Submit to a new window">
</form>
```

<input> height and width Attributes

The height and width attributes specify the height and width of an <input> element.

Note: The height and width attributes are only used with <input type="image">.

Tip: Always specify both the height and width attributes for images. If height and width are set, the space required for the image is reserved when the page is loaded. However, without these attributes, the browser does not know the size of the image, and cannot reserve the appropriate space to it. The effect will be that the page layout will change during loading (while the images load).

Example

Define an image as the submit button, with height and width attributes:

```
<input type="image" src="img_submit.gif" alt="Submit" width="48"
height="48">
```

<input> list Attribute

The list attribute refers to a <datalist> element that contains pre-defined options for an <input> element.

Example

An <input> element with pre-defined values in a <datalist>:

```
<input list="browsers">

<datalist id="browsers">
  <option value="Internet Explorer">
  <option value="Firefox">
  <option value="Chrome">
  <option value="Opera">
  <option value="Safari">
</datalist>
```

<input> min and max Attributes

The min and max attributes specify the minimum and maximum value for an <input> element.

Note: The min and max attributes works with the following input types: number, range, date, datetime, datetime-local, month, time and week.

Example

<input> elements with min and max values:

Enter a date before 1980-01-01:
<input type="date" name="bday" max="1979-12-31">

Enter a date after 2000-01-01:
<input type="date" name="bday" min="2000-01-02">

Quantity (between 1 and 5):
<input type="number" name="quantity" min="1" max="5">

<input> multiple Attribute

The multiple attribute is a boolean attribute.

When present, it specifies that the user is allowed to enter more than one value in the <input> element.

Note: The multiple attribute works with the following input types: email, and file.

Example

A file upload field that accepts multiple values:

Select images: <input type="file" name="img" multiple="multiple">

<input> pattern Attribute

The pattern attribute specifies a regular expression that the <input> element's value is checked against.

Note: The pattern attribute works with the following input types: text, search, url, tel, email, and password.

Tip: Use the global title attribute to describe the pattern to help the user.

Tip: Learn more about regular expressions in our JavaScript tutorial.

Example

An input field that can contain only three letters (no numbers or special characters):

Country code: <input type="text" name="country_code" pattern="[A-Za-z]{3}" title="Three letter country code">

<input> placeholder Attribute

The placeholder attribute specifies a short hint that describes the expected value of an input field (e.g. a sample value or a short description of the expected format).

The hint is displayed in the input field when it is empty, and disappears when the field gets focus.

Note: The placeholder attribute works with the following input types: text, search, url, tel, email, and password.

Example

An input field with a placeholder text:

<input type="text" name="fname" placeholder="First name">

<input> required Attribute

The required attribute is a boolean attribute.

When present, it specifies that an input field must be filled out before submitting the form.

Note: The required attribute works with the following input types: text, search, url, tel, email, password, date pickers, number, checkbox, radio, and file.

Example

A required input field:

Username: <input type="text" name="usrname" required="required">

<input> step Attribute

The step attribute specifies the legal number intervals for an <input> element.

Example: if step="3", legal numbers could be -3, 0, 3, 6, etc.

Tip: The step attribute can be used together with the max and min attributes to create a range of legal values.

Note: The step attribute works with the following input types: number, range, date, datetime, datetime-local, month, time and week.

Example

An input field with a specified legal number intervals:

<input type="number" name="points" step="3">

HTML5 Web Storage

HTML5 web storage is a better local storage than cookies.

With HTML5, web pages can store data locally within the user's browser.

Earlier, this was done with cookies. However, Web Storage is more secure and faster. The data is not included with every server request, but used ONLY when asked for. It is also possible to store large amounts of data, without affecting the website's performance.

The data is stored in key/value pairs, and a web page can only access data stored by itself.

Browser Support

Web storage is supported in Internet Explorer 8+, Firefox, Opera, Chrome, and Safari.

Note: Internet Explorer 7 and earlier versions, do not support web storage.

---

localStorage and sessionStorage

There are two new objects for storing data on the client:

- localStorage - stores data with no expiration date
- sessionStorage - stores data for one session

Before using web storage, check browser support for localStorage and sessionStorage:

```
if(typeof(Storage)!=="undefined")
 {
 // Yes! localStorage and sessionStorage support!
 // Some code.....
 }
else
 {
 // Sorry! No web storage support..
 }
```

The localStorage Object

The localStorage object stores the data with no expiration date. The data will not be deleted when the browser is closed, and will be available the next day, week, or year.

Example

```
localStorage.lastname="Smith";
document.getElementById("result").innerHTML="Last name: "
+ localStorage.lastname;
```

Example explained:

- Create a localStorage key/value pair with key="lastname" and value="Smith"
- Retrieve the value of the "lastname" key and insert it into the element with id="result"

Tip: Key/value pairs are always stored as strings. Remember to convert them to another format when needed.

The following example counts the number of times a user has clicked a button. In this code the value string is converted to a number to be able to increase the counter:

Example

```
if (localStorage.clickcount)
 {
 localStorage.clickcount=Number(localStorage.clickcount)+1;
 }
else
 {
 localStorage.clickcount=1;
 }
document.getElementById("result").innerHTML="You have clicked the button
" + localStorage.clickcount + " time(s).";
```

The sessionStorage Object

The sessionStorage object is equal to the localStorage object, **except** that it stores the data for only one session. The data is deleted when the user closes the browser window.

The following example counts the number of times a user has clicked a button, in the current session:

Example

```
if (sessionStorage.clickcount)
 {
 sessionStorage.clickcount=Number(sessionStorage.clickcount)+1;
 }
```

```
else
 {
 sessionStorage.clickcount=1;
 }
document.getElementById("result").innerHTML="You have clicked the button
" + sessionStorage.clickcount + " time(s) in this session.";
```

HTML5 Application Cache

With HTML5 it is easy to make an offline version of a web application, by creating a cache manifest file.

What is Application Cache?

HTML5 introduces application cache, which means that a web application is cached, and accessible without an internet connection.

Application cache gives an application three advantages:

1. Offline browsing - users can use the application when they're offline
2. Speed - cached resources load faster
3. Reduced server load - the browser will only download updated/changed resources from the server

---

Browser Support

Application cache is supported in all major browsers, except Internet Explorer.

HTML5 Cache Manifest Example

The example below shows an HTML document with a cache manifest (for offline browsing):

Example

```
<!DOCTYPE HTML>
<html manifest="demo.appcache">

<body>
The content of the document.....
</body>

</html>
```

Cache Manifest Basics

To enable application cache, include the manifest attribute in the document's <html> tag:

```
<!DOCTYPE HTML>
<html manifest="demo.appcache">
...
</html>
```

Every page with the manifest attribute specified will be cached when the user visits it. If the manifest attribute is not specified, the page will not be cached (unless the page is specified directly in the manifest file).

The recommended file extension for manifest files is: ".appcache"

A manifest file needs to be served with the **correct MIME-type**, which is "text/cache-manifest". Must be configured on the web server.

The Manifest File

The manifest file is a simple text file, which tells the browser what to cache (and what to never cache).

The manifest file has three sections:

- **CACHE MANIFEST** - Files listed under this header will be cached after they are downloaded for the first time
- **NETWORK** - Files listed under this header require a connection to the server, and will never be cached
- **FALLBACK** - Files listed under this header specifies fallback pages if a page is inaccessible

CACHE MANIFEST

The first line, CACHE MANIFEST, is required:

```
CACHE MANIFEST
/theme.css
/logo.gif
/main.js
```

The manifest file above lists three resources: a CSS file, a GIF image, and a JavaScript file. When the manifest file is loaded, the browser will download the three files from the root directory of the web site. Then, whenever the user is not connected to the internet, the resources will still be available.

NETWORK

The NETWORK section below specifies that the file "login.asp" should never be cached, and will not be available offline:

NETWORK:
login.asp

An asterisk can be used to indicate that all other resources/files require an internet connection:

NETWORK:
*

FALLBACK

The FALLBACK section below specifies that "offline.html" will be served in place of all files in the /html/ catalog, in case an internet connection cannot be established:

FALLBACK:
/html/ /offline.html

Note: The first URI is the resource, the second is the fallback.

Updating the Cache

Once an application is cached, it remains cached until one of the following happens:

- The user clears the browser's cache
- The manifest file is modified (see tip below)
- The application cache is programmatically updated

Example - Complete Cache Manifest File

CACHE MANIFEST
2012-02-21 v1.0.0
/theme.css
/logo.gif
/main.js

NETWORK:
login.asp

FALLBACK:
/html/ /offline.html

Tip: Lines starting with a "#" are comment lines, but can also serve another purpose. An application's cache is only updated when its manifest file changes. If you edit an image or change a JavaScript function, those changes will not be re-cached. Updating the date and version in a comment line is one way to make the browser re-cache your files.

Notes on Application Cache

Be careful with what you cache.

Once a file is cached, the browser will continue to show the cached version, even if you change the file on the server. To ensure the browser updates the cache, you need to change the manifest file.

Note: Browsers may have different size limits for cached data (some browsers have a 5MB limit per site).

HTML5 Web Workers

A web worker is a JavaScript running in the background, without affecting the performance of the page.

What is a Web Worker?

When executing scripts in an HTML page, the page becomes unresponsive until the script is finished.

A web worker is a JavaScript that runs in the background, independently of other scripts, without affecting the performance of the page. You can continue to do whatever you want: clicking, selecting things, etc., while the web worker runs in the background.

Browser Support

Web workers are supported in all major browsers, except Internet Explorer.

Check Web Worker Support

Before creating a web worker, check whether the user's browser supports it:

```
if(typeof(Worker)!=="undefined")
 {
 // Yes! Web worker support!
 // Some code.....
 }
else
 {
 // Sorry! No Web Worker support..
 }
```

Create a Web Worker File

Now, let's create our web worker in an external JavaScript.

Here, we create a script that counts. The script is stored in the "demo_workers.js" file:

```
var i=0;

function timedCount()
{
i=i+1;
postMessage(i);
setTimeout("timedCount()",500);
}

timedCount();
```

The important part of the code above is the **postMessage()** method - which is used to posts a message back to the HTML page.

Note: Normally web workers are not used for such simple scripts, but for more CPU intensive tasks.

Create a Web Worker Object

Now that we have the web worker file, we need to call it from an HTML page.

The following lines checks if the worker already exists, if not - it creates a new web worker object and runs the code in "demo_workers.js":

```
if(typeof(w)=="undefined")
 {
 w=new Worker("demo_workers.js");
 }
```

Then we can send and receive messages from the web worker.

Add an "onmessage" event listener to the web worker.

```
w.onmessage=function(event){
document.getElementById("result").innerHTML=event.data;
};
```

When the web worker posts a message, the code within the event listener is executed. The data from the web worker is stored in event.data.

Terminate a Web Worker

When a web worker object is created, it will continue to listen for messages (even after the external script is finished) until it is terminated.

To terminate a web worker, and free browser/computer resources, use the terminate() method:

```
w.terminate();
```

Full Web Worker Example Code

We have already seen the Worker code in the .js file. Below is the code for the HTML page:

Example

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

<p>Count numbers: <output id="result"></output></p>
<button onclick="startWorker()">Start Worker</button>
<button onclick="stopWorker()">Stop Worker</button>
<br><br>

<script>
var w;
```

```
function startWorker()
{
if(typeof(Worker)!=="undefined")
{
 if(typeof(w)=="undefined")
   {
   w=new Worker("demo_workers.js");
   }
 w.onmessage = function (event) {
   document.getElementById("result").innerHTML=event.data;
 };
}
else
{
document.getElementById("result").innerHTML="Sorry, your browser does
not support Web Workers...";
}
}

function stopWorker()
{
w.terminate();
}
</script>

</body>
</html>
```

Web Workers and the DOM

Since web workers are in external files, they do not have access to the following JavaScript objects:

- The window object
- The document object
- The parent object

HTML5 Server-Sent Events

HTML5 Server-Sent Events allow a web page to get updates from a server.

Server-Sent Events - One Way Messaging

A server-sent event is when a web page automatically gets updates from a server.

This was also possible before, but the web page would have to ask if any updates were available. With server-sent events, the updates come automatically.

Examples: Facebook/Twitter updates, stock price updates, news feeds, sport results, etc.

Browser Support

Server-Sent Events are supported in all major browsers, except Internet Explorer.

Receive Server-Sent Event Notifications

The EventSource object is used to receive server-sent event notifications:

Example

```
var source=new EventSource("demo_sse.php");
source.onmessage=function(event)
 {
 document.getElementById("result").innerHTML+=event.data + "<br>";
 };
```

Example explained:

- Create a new EventSource object, and specify the URL of the page sending the updates (in this example "demo_sse.php")
- Each time an update is received, the onmessage event occurs
- When an onmessage event occurs, put the received data into the element with id="result"

Check Server-Sent Events Support

In the tryit example above there were some extra lines of code to check browser support for server-sent events:

```
if(typeof(EventSource)!=="undefined")
 {
 // Yes! Server-sent events support!
 // Some code.....
 }
else
 {
 // Sorry! No server-sent events support..
 }
```

Server-Side Code Example

For the example above to work, you need a server capable of sending data updates (like PHP or ASP).

The server-side event stream syntax is simple. Set the "Content-Type" header to "text/event-stream". Now you can start sending event streams.

Code in PHP (demo_sse.php):

```
<?php
header('Content-Type: text/event-stream');
header('Cache-Control: no-cache');

$time = date('r');
echo "data: The server time is: {$time}\n\n";
flush();
?>
```

Code in ASP (VB) (demo_sse.asp):

```
<%
Response.ContentType="text/event-stream"
Response.Expires=-1
Response.Write("data: " & now())
Response.Flush()
%>
```

Code explained:

- Set the "Content-Type" header to "text/event-stream"
- Specify that the page should not cache
- Output the data to send (**Always** start with "data: ")
- Flush the output data back to the web page

The EventSource Object

In the examples above we used the onmessage event to get messages. But other events are also available:

Events	Description
onopen	When a connection to the server is opened
onmessage	When a message is received
onerror	When an error occurs

HTML5 Canvas

The <canvas> element is used to draw graphics, on the fly, on a web page.

Draw a red rectangle, a gradient rectangle, a multicolor rectangle, and some multicolor text onto the canvas:

What is Canvas?

The HTML5 <canvas> element is used to draw graphics, on the fly, via scripting (usually JavaScript).

The <canvas> element is only a container for graphics. You must use a script to actually draw the graphics.

Canvas has several methods for drawing paths, boxes, circles, characters, and adding images.

Browser Support

Internet Explorer 9, Firefox, Opera, Chrome, and Safari support the <canvas> element.

Note: Internet Explorer 8 and earlier versions, do not support the <canvas> element.

Create a Canvas

A canvas is a rectangular area on an HTML page, and it is specified with the <canvas> element.

Note: By default, the <canvas> element has no border and no content.

The markup looks like this:

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

Note: Always specify an id attribute (to be referred to in a script), and a width and height attribute to define the size of the canvas.

Tip: You can have multiple <canvas> elements on one HTML page.

To add a border, use the style attribute:

Example

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

Draw Onto The Canvas With JavaScript

All drawing on the canvas must be done inside a JavaScript:

Example

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

Example explained:

First, find the <canvas> element:

```
var c=document.getElementById("myCanvas");
```

Then, call its getContext() method (you must pass the string "2d" to the getContext() method):

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

The getContext("2d") object is a built-in HTML5 object, with many properties and methods for drawing paths, boxes, circles, text, images, and more.

The next two lines draws a red rectangle:

```
ctx.fillStyle="#FF0000";
ctx.fillRect(0,0,150,75);
```

The fillStyle property can be a CSS color, a gradient, or a pattern. The default fillStyle is #000000 (black).

The fillRect(*x,y,width,height*) method draws a rectangle filled with the current fill style.

Canvas Coordinates

The canvas is a two-dimensional grid.

The upper-left corner of the canvas has coordinate (0,0)

So, the fillRect() method above had the parameters (0,0,150,75).

This means: Start at the upper-left corner (0,0) and draw a 150x75 pixels rectangle.

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(300,150);
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().

Example

Create a circle with the arc() method:

JavaScript:

```
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)

Using fillText():

Example

Write a 30px high filled text on the canvas, using the font "Arial":

JavaScript:

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

```
ctx.font="30px Arial";
ctx.fillText("Hello World",10,50);
```

Using strokeText():

Example

Write a 30px high text (no fill) on the canvas, using the font "Arial":

JavaScript:

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

Canvas - Gradients

Gradients can be used to fill rectangles, circles, lines, text, etc. Shapes on the canvas are not limited to solid colors.

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

Once we have a gradient object, we must add two or more color stops.

The addColorStop() method specifies the color stops, and its position along the gradient. Gradient positions can be anywhere between 0 to 1.

To use the gradient, set the fillStyle or strokeStyle property to the gradient, and then draw the shape, like a rectangle, text, or a line.

Using createLinearGradient():

Example

Create a linear gradient. Fill rectangle with the gradient:

JavaScript:

```
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():

Example

Create a radial/circular gradient. Fill rectangle with the gradient:

JavaScript:

```
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*)

Image to use:

Example

Draw the image onto the canvas:

JavaScript:

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

HTML canvas arc() Method

Example

Create a circle:

JavaScript:

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.beginPath();
ctx.arc(100,75,50,0,2*Math.PI);
ctx.stroke();
```

Browser Support

Internet Explorer 9, Firefox, Opera, Chrome, and Safari support the arc() method.

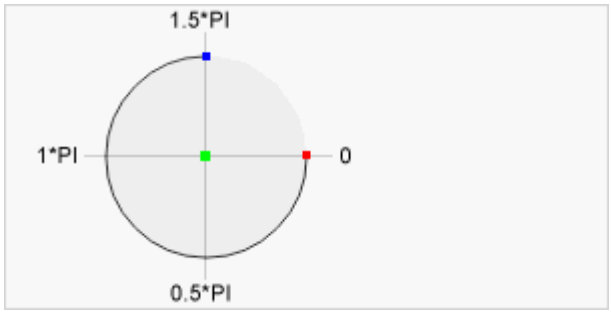
Note: Internet Explorer 8 and earlier versions, do not support the <canvas> element.

Definition and Usage

The arc() method creates an arc/curve (used to create circles, or parts of circles).

Tip: To create a circle with arc(): Set start angle to 0 and end angle to 2*Math.PI.

Tip: Use the stroke() or the fill() method to actually draw the arc on the canvas.



Center

arc(**100,75**,50,0*Math.PI,1.5*Math.PI)

Start angle

arc(100,75,50,**0**,1.5*Math.PI)

End angle

arc(100,75,50,0*Math.PI,**1.5*Math.PI**)

JavaScript syntax: *context*.arc(*x,y,r,sAngle,eAngle*);

Parameter Values

Parameter	Description
x	The x-coordinate of the center of the circle
y	The y-coordinate of the center of the circle
r	The radius of the circle
sAngle	The starting angle, in radians (0 is at the 3 o'clock position of the arc's circle)
eAngle	The ending angle, in radians
counterclockwise	Optional. Specifies whether the drawing should be counterclockwise or clockwise. False=clockwise, true=counter-clockwise

HTML canvas quadraticCurveTo() Method

Example

Draw a quadratic Bézier curve:

JavaScript:

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.beginPath();
ctx.moveTo(20,20);
ctx.quadraticCurveTo(20,100,200,20);
ctx.stroke();
```

Browser Support

Internet Explorer 9, Firefox, Opera, Chrome, and Safari support the quadraticCurveTo() method.

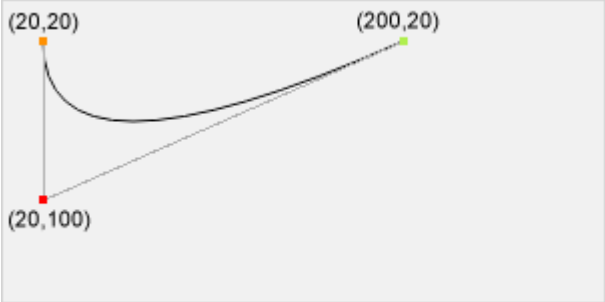
Note: Internet Explorer 8 and earlier versions, do not support the <canvas> element.

Definition and Usage

The quadraticCurveTo() method adds a point to the current path by using the specified control points that represent a quadratic Bézier curve.

A quadratic Bézier curve requires two points. The first point is a control point that is used in the quadratic Bézier calculation and the second point is the ending point for the curve. The starting point for the curve is the last point in

the current path. If a path does not exist, use the beginPath() and moveTo() methods to define a starting point.



Start point:
moveTo(**20,20**)

Control point:

quadraticCurveTo(**20,100**,200,20)

End point:

quadraticCurveTo(20,100,**200,20**)

Tip: Check out the bezierCurveTo() method. It has two control-points instead of one.

JavaScript syntax: *context*.quadraticCurveTo(*cpx*,*cpy*,*x*,*y*);

Parameter Values

Parameter	Description	Play it
cpx	The x-coordinate of the Bézier control point	Play it »
cpy	The y-coordinate of the Bézier control point	Play it »
x	The x-coordinate of the ending point	Play it »
y	The y-coordinate of the ending point	Play it »

HTML canvas bezierCurveTo() Method

Example

Draw a cubic Bézier curve:

JavaScript:

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.beginPath();
ctx.moveTo(20,20);
ctx.bezierCurveTo(20,100,200,100,200,20);
ctx.stroke();
```

Browser Support

Internet Explorer 9, Firefox, Opera, Chrome, and Safari support the bezierCurveTo() method.

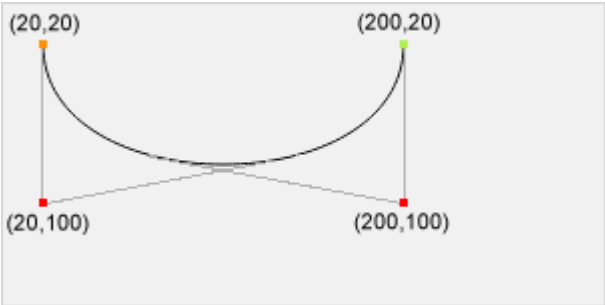
Note: Internet Explorer 8 and earlier versions, do not support the <canvas> element.

Definition and Usage

The bezierCurveTo() method adds a point to the current path by using the specified control points that represent a cubic Bézier curve.

A cubic bezier curve requires three points. The first two points are control points that are used in the cubic Bézier calculation and the last point is the ending point for the curve. The starting point for the curve is the last point in

the current path. If a path does not exist, use the beginPath() and moveTo() methods to define a starting point.



Start point

moveTo(**20,20**)

Control point 1

bezierCurveTo(**20,100**,200,100,200,20)

Control point 2

bezierCurveTo(20,100,**200,100**,200,20)

End point

bezierCurveTo(20,100,200,100,**200,20**)

Tip: Check out the quadraticCurveTo() method. It has one control point instead of two.

JavaScript syntax: *context*.bezierCurveTo(*cp1x,cp1y,cp2x,cp2y,x,y*);

Parameter Values

Parameter	Description	Play it
cp1x	The x-coordinate of the first Bézier control point	Play it »
cp1y	The y-coordinate of the first Bézier control point	Play it »
cp2x	The x-coordinate of the second Bézier control point	Play it »
cp2y	The y-coordinate of the second Bézier control point	Play it »
x	The x-coordinate of the ending point	Play it »
y	The y-coordinate of the ending point	Play it »

HTML canvas createLinearGradient() Method

Example

Define a gradient (left to right) that goes from black to white, as the fill style for the rectangle:

JavaScript:

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

var grd=ctx.createLinearGradient(0,0,170,0);
grd.addColorStop(0,"black");
grd.addColorStop(1,"white");

ctx.fillStyle=grd;
ctx.fillRect(20,20,150,100);
```

Browser Support

Internet Explorer 9, Firefox, Opera, Chrome, and Safari support the createLinearGradient() method.

Note: Internet Explorer 8 and earlier versions, do not support the <canvas> element.

Definition and Usage

The createLinearGradient() method creates a linear gradient object.

The gradient can be used to fill rectangles, circles, lines, text, etc.

Tip: Use this object as the value to the strokeStyle or fillStyle properties.

Tip: Use the addColorStop() method to specify different colors, and where to position the colors in the gradient object.

JavaScript syntax: *context*.createLinearGradient(*x0,y0,x1,y1*);

Parameter Values

	Description
x0	The x-coordinate of the start point of the gradient
y0	The y-coordinate of the start point of the gradient
x1	The x-coordinate of the end point of the gradient
y1	The y-coordinate of the end point of the gradient

More Examples

Example

Define a gradient (top to bottom) as the fill style for the rectangle:

JavaScript:

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
var my_gradient=ctx.createLinearGradient(0,0,0,170);
my_gradient.addColorStop(0,"black");
my_gradient.addColorStop(1,"white");
ctx.fillStyle=my_gradient;
ctx.fillRect(20,20,150,100);
```

Example

Define a gradient that goes from black, to red, to white, as the fill style for the rectangle:

JavaScript:

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
var my_gradient=ctx.createLinearGradient(0,0,170,0);
my_gradient.addColorStop(0,"black");
my_gradient.addColorStop(0.5,"red");
my_gradient.addColorStop(1,"white");
ctx.fillStyle=my_gradient;
ctx.fillRect(20,20,150,100);
```

HTML5 Inline SVG

HTML5 has support for 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)

Browser Support

Internet Explorer 9, Firefox, Opera, Chrome, and Safari support inline SVG.

Embed SVG Directly Into HTML Pages

In HTML5, you can embed SVG elements directly into your HTML page:

Example

```
<!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.

Comparison of Canvas and SVG

The table below shows some important differences between Canvas and SVG:

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