purpose of the attribute is not to supply instructions for the field’s use but rather to supply

a potential value for the user to submit to the server.

HTML5 introduces the **placeholder**

attribute to use instead for this duty:

**<input type="text" name="firstname" id="firstname"**

**placeholder="Enter your name here">**

HTML5 also introduces the **autofocus** attribute, which when placed on a field should

cause a supporting browser to immediately focus this field once the page is loaded:

**<label>**Search:**<input type="search" name="query"**

**id="searchBox" autofocus></label>**

Also under HTML5, it should be possible to advise the browser to display the

**autocomplete** suggestions provided for fields if similar field names have been used in the

past:

**<input type="text" name="firstname" id="firstname"**

**placeholder="Enter your name here" autocomplete>**

Interestingly, this particular attribute has been supported in Internet Explorer

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

HTML5 Event Attributes

onabort, onblur, oncanplay, oncanplaythrough, onchange, onclick,

oncontextmenu, ondblclick, ondrag, ondragend, ondragenter, ondragleave,

ondragover, ondragstart, ondrop, ondurationchange, onemptied, onended,

onerror, onfocus, onformchange, onforminput, oninput, oninvalid, onkeydown,

onkeypress, onkeyup, onload, onloadeddata, onloadedmetadata, onloadstart,

onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onmousewheel,

onpause, onplay, onplaying, onprogress, onratechange, onreadystatechange,

onscroll, onseeked, onseeking, onselect, onshow, onstalled, onsubmit,

onsuspend, ontimeupdate, onvolumechange, onwaiting

Element-Specific Attributes

media This attribute defines the intended media type of the linked media source, to

provide a hint to a user agent as to whether the media referenced is appropriate or how it

might be used. It is similar to the idea of a **media** attribute in a style sheet specifying **print**,

**screen**, **projection**, or other common values.

src This attribute is set to the URL of the media source to link to.

type This attribute is set to the MIME type of the linked media file specified by the **src**

attribute. Often it also includes a codecs value to indicate how a media resource is

encoded. However, the use of codecs, as alluded to in Chapter 2, is a bit of a mess under

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

Text between the <audio> and </audio> tags will display in browsers that do not support the <audio> element.

Multiple **<source>** elements can link to different audio files. The browser will use the first recognized format.

HTML5 so page authors are urged to test carefully.

Examples

<!-- Multiple sources to try -->

**<audio>**

**<source src="angus.ogg">**

**<source src="angus.mp4" type="audio/mp4">**

**</audio>**

<!-- XHTML style -->

**<video>**

**<source src="angus.mp4" type="video/mp4; codecs='avc1.58A01E, mp4a.40.2'" />**

**</video>**

Compatibility

HTML5 Firefox 3.5+, Safari 3.1+

HTML Audio - Browser Support

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Browser** | **MP3** | **Wav** | **Ogg** |
| Internet Explorer | YES | NO | NO |
| Chrome | YES | YES | YES |
| Firefox | YES | YES | YES |
| Safari | YES | YES | NO |
| Opera | YES | YES | YES |

HTML Audio - Media Types

|  |  |
| --- | --- |
| **File Format** | **Media Type** |
| MP3 | audio/mpeg |
| Ogg | audio/ogg |
| Wav | audio/wav |

## Definition and Usage

The preload attribute specifies if and how the author thinks that the video should be loaded when the page loads.

The preload attribute allows the author to provide a hint to the browser about what he/she thinks will lead to the best user experience. This attribute may be ignored in some instances.

**Note:** The preload attribute is ignored if autoplay is present.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

Article

## Definition and Usage

The <article> tag specifies independent, self-contained content.

An article should make sense on its own and it should be possible to distribute it independently from the rest of the site.

Potential sources for the <article> element:

* Forum post
* Blog post
* News story
* Comment

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Modernizer\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Modernizr is a JavaScript library (found at http://modernizr.com) that detects what

HTML5 and CSS3 features the browser supports and makes it simple for developers

to test and code for the browsers that do not support some of the new technologies. In

the previous examples, we used standard JavaScript to test for features, but with Modernizr

it is incredibly easy.

Although Modernizr does the detection part for you, it does *not*, however, fill in

the gaps and add the missing functionality for you. Regardless, it is an incredibly powerful

bit of script that developers should be aware of and have ready in their arsenal.

Now on version 2, the Modernizr library focuses on CSS3 detection, HTML5 tag

support, and JavaScript API support. For each of these areas, the Modernizr library has

particular properties that can be accessed after initialization and used to dynamically

change the source and thus support multiple experiences and browsers. Some of the

property f lags include the following:

n Geolocation API

n localStorage

n sessionStorage

n Drag and Drop

n History Management

n applicationCache

The big question is: What can we do for legacy browsers?

The answer is that you don’t retire your pre-existing JavaScript

validation just yet, but you leave it as a fallback after doing

some feature detection. For instance, to detect whether

<input type=email> is supported, you make a new

<input type=email> with JavaScript, but don’t add it to the page.

Then, you interrogate your new element to fi nd out what its type

attribute is. If it’s reported back as “email”, then the browser

supports the new feature—so let it do its work and don’t bring

in any JavaScript validation. If it’s reported back as “text”, it’s

fallen back to the default, indicating that it’s not supported. So

your code should load in an alternative validation library ideally

through a lazy load technique so that by default, HTML5-aware

browsers don’t need to download and process JavaScript that

isn’t required.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Canvas\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The hello world of any canvas demo starts with putting the

canvas element on your page. Initially the canvas is completely

invisible and by default is 300 pixels wide by 150 pixels high:

The HTML **<canvas>** element (introduced in HTML5) is a **container** for canvas graphics.

An HTML canvas is a rectangular area on an HTML page.

Canvas has several methods for drawing paths, boxes, circles, text, and graphic images

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CSS Selector\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***