HTML5

Syntax Reference Guide

HTML5 is the most recent markup language in the HTML lineage, used to create and present content over the World Wide Web, with the intent of communicating information to a targeted populous of web users.

HTML uses a tree structure for elements, which nests any number elements within a single element. Mostly elements have one parent, but can easily several or more.

When authoring a mark-up document, two types of syntax can be employed: HTML and XHTML. HTML allows flexibility in design and handling requirements. But, XHTML has more rigorous and stringent syntax requirements, as it is based on XML. XML is a structure set of rules that allows one to define how to organize data, hence the name Extensible Markup Language; it allows for tailoring the design for a specific purpose. This concept precipitated the emergence of having this same type adaptation over web pages, and so XHTML was born (XML + HTML).

Let's look at some specific items used to create a markup language:

Attributes

- Standard Attributes
- Element Specific
- Global Attributes
- accesskey
- class
- contenteditable
- data
- dir
- draggable
- dropzone
- hidden
- id
- lang
- spellcheck
- style
- tabindex
- title
- translate

Basic **HTML** Example:

```
<!DOCTYPE html>
<html lang=”en”>
 <head>
   <title>My HTML Document</title>
 </head>
 <body class=example>
   <h1>Subject</h1>
   <p>My HTML document.</p>
 </body>
</html>
```

Let's go through our simple example, and explain a few things.

HTML Tag

The <html> start and end tags are actually optional in the HTML document, and can be safely omitted if need be. Additionally, the lang attribute in the html element tag is optional. But it's considered good practice to include the specified language of your document.

Head Tag

Metadata is contained within our <head></head> tags. Metadata is simply information about our document. That information can be any of the following nested elements in our <head> element: <title>, <style>, <base>, <link>, <meta>, <script>, and <noscript>, or any of our global attributes. Primarily, what you will see is just the <title> element being used. However, with HTML5, the <head> element can be omitted altogether, and the <title> element can just be used. Either way, <title> is a mandatory tag within our <html>…</html> tags. The document must have a title/name. Similar to <head>, the <title> element supports global attributes.

The <base> element also produces metadata, in that it defines a default URL and target for all links on our web page, specifying href, and target attributes.

An important element not shown in our example is the <link> element. This element links resources like CSS to our document. It supports global attributes, as well as the following standard: href, rel, media, hreflang, type, and sizes.

The <meta> element is essentially data information about data. For example, it gives the character set/character encoding to use in our document, that the browser may interpret it. It supports global attributes, as well as the following attributes: name, http-equiv, content, and charset.

The <style> element allows for the aesthetic manipulation of a webpage (background color, font size, font color, etc.). It supports the global attributes, as well as the following standard attributes: media, type, and scoped.

The <script> element permits scripting languages like JavaScript to be included in your HTML document. Although, traditionally the <script> tag is within the <head>…</head> elements, HTML5 supports it being either there, or within the <body>…</body> tags. It supports the global attributes, as well as the following standard attributes: src, async, defer, type, and charset.

The <noscript> element defines content for browsers that do not support scripting languages or have scripts disabled. The content within the <noscript>…</noscript> tags will be displayed if scripts are not supported. The <noscript> only supports the global attributes.

Body Tag

The <body>…</body> element contains all of the main content of the HTML document. The global attributes are supported, as well as the following standard attributes: onbeforeunload, onerror, onhashchange, onload, onmessage, onoffline, ononline, onpopstate, onresize, onstorage, onunload.

H1-H6 Tags

The <h1>…</h1> tags are one set of six tags (<h1> - <h6>), all varying in degree of font size, with the <h1> being the largest. As the number increments the font size is progressively smaller. The h1-h6 tags only support the global attributes.

P Tag

As the name suggests, the … tag define paragraphs in an HTML document, single line or multi-line. This element only support the global attributes.

That covers our example. Let's compare our HTML with XHTML.

Basic **XHTML** Example:

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
 <head>
   <title>My HTML Document</title>
 </head>
 <body class="example">
   <h1>Subject</h1>
   <p>My HTML document.</p>
 </body>
</html>
```

Because the strictness of the language, the <html> start and end tags are mandatory in XHTML. And similar to our HTML example, the lang attribute in the html element tag is optional. But, it's good practice…. The syntax is slightly different in that it does not require the <!DOCTYPE html> tag (the browser interprets the XHTML as XML), and in the lang element syntax(it is also legal to just use lang="en").

XHTML requires that start and close tags are used, as it adheres to strict coding rules, based on XML. XHTML documents may appear more verbose than HTML. But, the readability will be about the same.

Due to the similarities of both the HTML and XHTML syntaxes, it is possible to markup documents using a common subset of the syntax that is the same in both, while avoiding the syntactic features that are unique to each. This type of document is called polyglot markup, or more simply: a polyglot document. It conforms to both syntaxes and treated favorably as either.

Tags

All elements are identified by their tag name and are marked up using either start tags, end tags, or self-closing tags. A start tag marks the beginning of an element, while an end tag marks the end. Start tags are delimited using angle brackets with the tag name and any attributes in between. End tags are delimited by angle brackets with a slash before the tag name.

A self-closing tag is one that has a slash immediately before the closing right angle bracket. This indicates that the element is to be closed immediately, and that it has no content. When a Self-closing tag is used, the end tag is omitted. In HTML, Self-closing tags are used by void and foreign elements.

Self-closing tag syntax (
 tag):

```
<p>Merry Christmas,<br/>
and Happy New Year!</p>
```


A container that manages other page elements and segments our HTML document. It's mostly used as a grouping element.

```
<div class=”cname”>
    <img src=”images/myimage.jpg” alt=”myname” height=”28”
width=”28”>
</div>
```


The span tag is a kind of place holder for text, where it can style text color, size, and action (eg. timer).

```
<span id=”tracktime” style=”color:#0000ff; font-weight:bold” >
00:00 / 00:00  </span>
```

<audio>

The audio tag is a tag supported in HTML5 that allows native audio support.

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

<video>

The video tag is a tag supported in HTML5 that allows native video support.

```
<video width=”320” height=”240”controls>
  <source src=”MyVideo.mp4” type=”video/mp4”>
  <source src=”MyVideo.ogg” type=”video/ogg”>
Your browser does not support the video element.
</video>
```

<script>

External File: `<script src=”js/playlist.js”></script>`
Internal JS code: `<script>…</script>`

Elements

Void elements - represent elements that are empty: eg. br, hr, link, meta

XHTML syntax requires an explicit end tag:
`<hr></hr>`

End tags are not permitted for void elements in the HTML syntax:
`<hr/>`

Raw Text Elements

Raw text elements support content that is treated as raw text instead of markup. Start and end tags are both needed for this type element. Additionally, the content cannot match the closing bracket syntax (/>). If so, the content (raw text) will need to be escaped with “\”

Example:

The javascript can be modified by escaping the slash, a work around for the HTML syntax error.

```
<script>
if (…) {
        document.write(“<script src=\”example.js\”><\/script>”);
}
</script>
```

RCDATA Elements

These are elements are such that they support character references, but all other content is treated as raw text instead of markup. They can only contain text or character references.

```
<textarea>
   Lost in space with the moon &amp; Haley&apos;s comet, &lt; and &gt;,
</textarea>
```

Foreign Elements

Foreign elements refers to elements that are in the SVG and MathML namespaces. They require a start and end tag. Additionally, a self-closing tag may be used, where legal. These elements can contain text, character references, CDATA sections, other elements, and comments, but the text must not contain the less-than sign character (<) or an ambiguous ampersand.

Normal Elements

These are elements that have a start and end tag. However, they do not support self-closing tags. Normal elements can contain text, character references, other elements, and comments, but the text must not contain the less-than sign character (<) or an ambiguous ampersand.

Attributes

Elements may have attributes that are used to specify additional information about them. Some attributes are defined globally and can be used on any HTML element, while others are defined for specific elements only. Every attribute must have an attribute name that is used to identify it. Every attribute also has an associated attribute value, which, depending on the attribute's definition, may represent one of several different types. The permitted syntax for each attribute depends on the given value.

A typical attribute in HTML has a name and a value separated by an equals sign (=). Attributes are placed within a start tag and are separated from the tag name and from each other by whitespace. They must not be specified within an end tag.

Element attributes are surrounded by double or single quotes: <img src = “example.jpg”>, <img src = ‘example.jpg’>. The quotes cannot be mixed, either use double or single. If the attribute value itself includes a single or double quote, common sense should kick in and tell you to use the opposite type of quotes to quote the attribute value. e.g. <img src=”space.jpg” title=”Haley’s Comet”>. Or, simply use a reference character reference <img src=”space.jpg” title=”Haley's Comet”>

Mime Types

Simply speaking, a mime type is a media type, a format if you will. It consist of a type and subtype (type/subtype). An example would be, if we were dealing with audio, an mp3 mime type is audio (the base of what we are dealing with), and mpeg the specific type of audio (mime: audio/mpeg). This doesn't just stop at audio or video, but translates to other file types. Say we are designing a web page, we have to use text (type) to do that, the base of what we are working with. However, the tried and proven way to accomplish this is with HTML (subtype). So, we could say that the Mime for designing our basic web page is text/html. You get the gist.

Polyglot Document

A polyglot HTML document is a document that conforms to both the HTML and XHTML syntactic requirements, and which can be processed as either by browsers, depending on the MIME type used. This works by using a common subset of the syntax that is shared by both HTML and XHTML.

Polyglot documents are useful to create for situations where a document is intended to be served as either HTML or XHTML, depending on the support in particular browsers, or when it is not known at the time of creation, which MIME type the document will ultimately be served as.

In order to successfully create and maintain polyglot documents, authors need to be familiar with both the similarities and differences between the two syntaxes. This includes not only syntactic differences, but also differences in the way stylesheets, and scripts are handled, and the way in which character encodings are detected.

Metadata Content

Metadata content includes elements for marking up document metadata; marking up or linking to resources that describe the behavior or presentation of the document; or indicate relationships with other documents.

Metadata elements appear within the head of a document. Some common examples of metadata elements include: title, meta, link, script and style.

Sectioning Root

These elements can have their own outlines, but the sections and headers inside these elements do not contribute to the outlines of their ancestors.

Some common sectioning root elements include, among others, body, blockquote and figure.

Sectioning Content

Sectioning content (also known as flow content) is used for structuring a document into sections, each of which generally has its own heading. These elements provide a scope within which associated headers, footers and contact information apply.

Some common sectioning elements include, among others, section, article and nav.

Phrasing Content

Phrasing content includes text and text-level markup. This is similar to the concept of inline level elements in HTML 4.01. Most elements that are categorized as phrasing content can only contain other phrasing content.

Some common examples of phrasing content elements include abbr, em, strong and span.

Embedded Content

Embedded content includes elements that load external resources into the document. Such external resources include, for example, images, videos and Flash-based content. Some embedded content elements include img, object, embed and video.

Interactive Content

Interactive elements are those that allow the user to interact with or activate in some way. Depending on the user's browser and device, this could be performed using any kind of input device, such as, for example, a mouse, keyboard, touch screen or voice input.

Some common examples of interactive content include audio and video.

Transparent Content Models

Some elements have transparent content models, meaning that their allowed content depends upon the parent element. They may contain any content that their parent element may contain, in addition to any other allowances or exceptions described for the element.

Attributes

While standard attributes are contextual in nature to the elements they complement, global attributes can be used with any element throughout the document. They help to give an element identification, context, and new meaning. According to the World Wide Consortium's site, figure 1.0 annotates the most recent list of global attributes.

HTML Global Attributes

= Attribute added in HTML5.

Attribute	Description
accesskey	Specifies a shortcut key to activate/focus an element
class	Specifies one or more classnames for an element (refers to a class in a style sheet)
contenteditable	Specifies whether the content of an element is editable or not
contextmenu	Specifies a context menu for an element. The context menu appears when a user right-clicks on the element
data-*	Used to store custom data private to the page or application
dir	Specifies the text direction for the content in an element
draggable	Specifies whether an element is draggable or not
dropzone	Specifies whether the dragged data is copied, moved, or linked, when dropped
hidden	Specifies that an element is not yet, or is no longer, relevant
id	Specifies a unique id for an element
lang	Specifies the language of the element's content
spellcheck	Specifies whether the element is to have its spelling and grammar checked or not
style	Specifies an inline CSS style for an element
tabindex	Specifies the tabbing order of an element
title	Specifies extra information about an element
translate	Specifies whether the content of an element should be translated or not

Figure 1.0: W3C Global Attributes List

Events

Attribute	Value	Description
onabort	*script*	Script to be run on abort
oncanplay	*script*	Script to be run when a file is ready to start playing (when it has buffered enough to begin)
oncanplaythrough	*script*	Script to be run when a file can be played all the way to the end without pausing for buffering
oncuechange	*script*	Script to be run when the cue changes in a <track> element
ondurationchange	*script*	Script to be run when the length of the media changes
onemptied	*script*	Script to be run when something bad happens and the file is suddenly unavailable (like unexpectedly disconnects)
onended	*script*	Script to be run when the media has reach the end (a useful event for messages like "thanks for listening")
onerror	*script*	Script to be run when an error occurs when the file is being loaded
onloadeddata	*script*	Script to be run when media data is loaded
onloadedmetadata	*script*	Script to be run when meta data (like dimensions and duration) are loaded
onloadstart	*script*	Script to be run just as the file begins to load before anything is actually loaded
onpause	*script*	Script to be run when the media is paused either by the user or programmatically
onplay	*script*	Script to be run when the media is ready to start playing
onplaying	*script*	Script to be run when the media actually has started playing
onprogress	*script*	Script to be run when the browser is in the process of getting the media data
onratechange	*script*	Script to be run each time the playback rate changes (like when a user switches to a slow motion or fast forward mode)
onseeked	*script*	Script to be run when the seeking attribute is set to false indicating that seeking has ended
onseeking	*script*	Script to be run when the seeking attribute is set to true indicating that seeking is active
onstalled	*script*	Script to be run when the browser is unable to fetch the media data for whatever reason
onsuspend	*script*	Script to be run when fetching the media data is stopped before it is completely loaded for whatever reason
ontimeupdate	*script*	Script to be run when the playing position has changed (like when the user fast forwards to a different point in the media)
onvolumechange	*script*	Script to be run each time the volume is changed which (includes setting the volume to "mute")
onwaiting	*script*	Script to be run when the media has paused but is expected to resume (like when the media pauses to buffer more data)

Figure 1.1: Media Events

Window Event Attributes

Events triggered for the window object (applies to the <body> tag):

Attribute	Value	Description
onafterprint	*script*	Script to be run after the document is printed
onbeforeprint	*script*	Script to be run before the document is printed
onbeforeunload	*script*	Script to be run when the document is about to be unloaded
onerror	*script*	Script to be run when an error occurs
onhashchange	*script*	Script to be run when there has been changes to the anchor part of the a URL
onload	*script*	Fires after the page is finished loading
onmessage	*script*	Script to be run when the message is triggered
onoffline	*script*	Script to be run when the browser starts to work offline
ononline	*script*	Script to be run when the browser starts to work online
onpagehide	*script*	Script to be run when a user navigates away from a page
onpageshow	*script*	Script to be run when a user navigates to a page
onpopstate	*script*	Script to be run when the window's history changes
onresize	*script*	Fires when the browser window is resized
onstorage	*script*	Script to be run when a Web Storage area is updated
onunload	*script*	Fires once a page has unloaded (or the browser window has been closed)

Figure 1.2: Window Event Attributes

Form Events

Events triggered by actions inside a HTML form (applies to almost all HTML elements, but is most used in form elements):

Attribute	Value	Description
onblur	*script*	Fires the moment that the element loses focus
onchange	*script*	Fires the moment when the value of the element is changed
oncontextmenu	*script*	Script to be run when a context menu is triggered
onfocus	*script*	Fires the moment when the element gets focus
oninput	*script*	Script to be run when an element gets user input
oninvalid	*script*	Script to be run when an element is invalid
onreset	*script*	Fires when the Reset button in a form is clicked
onsearch	*script*	Fires when the user writes something in a search field (for <input="search">)
onselect	*script*	Fires after some text has been selected in an element
onsubmit	*script*	Fires when a form is submitted

Figure 1.3: Form Events

Keyboard Events

Attribute	Value	Description
onkeydown	*script*	Fires when a user is pressing a key
onkeypress	*script*	Fires when a user presses a key
onkeyup	*script*	Fires when a user releases a key

Figure 1.4: Keyboard Events

Mouse Events

Events triggered by a mouse, or similar user actions:

Attribute	Value	Description
onclick	*script*	Fires on a mouse click on the element
ondblclick	*script*	Fires on a mouse double-click on the element
ondrag	*script*	Script to be run when an element is dragged
ondragend	*script*	Script to be run at the end of a drag operation
ondragenter	*script*	Script to be run when an element has been dragged to a valid drop target
ondragleave	*script*	Script to be run when an element leaves a valid drop target
ondragover	*script*	Script to be run when an element is being dragged over a valid drop target
ondragstart	*script*	Script to be run at the start of a drag operation
ondrop	*script*	Script to be run when dragged element is being dropped
onmousedown	*script*	Fires when a mouse button is pressed down on an element
onmousemove	*script*	Fires when the mouse pointer is moving while it is over an element
onmouseout	*script*	Fires when the mouse pointer moves out of an element
onmouseover	*script*	Fires when the mouse pointer moves over an element
onmouseup	*script*	Fires when a mouse button is released over an element
onmousewheel	*script*	Deprecated. Use the onwheel attribute instead
onscroll	*script*	Script to be run when an element's scrollbar is being scrolled
onwheel	*script*	Fires when the mouse wheel rolls up or down over an element

Figure 1.5: Mouse Events

Clipboard Events

Attribute	Value	Description
oncopy	*script*	Fires when the user copies the content of an element
oncut	*script*	Fires when the user cuts the content of an element
onpaste	*script*	Fires when the user pastes some content in an element

Figure 1.6: Clipboard Events