

HTML5

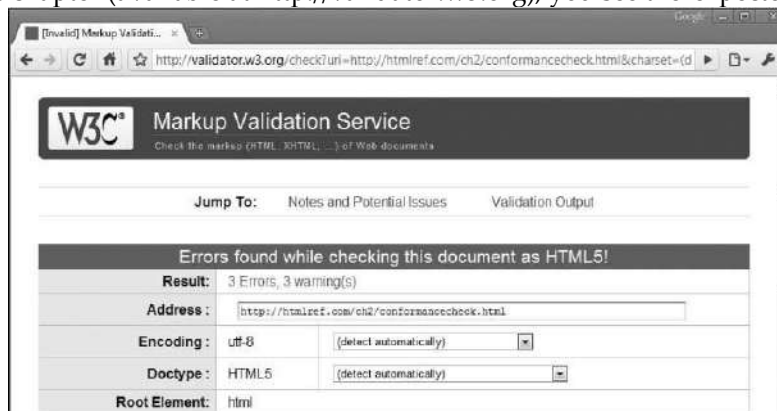
Hello HTML5

The syntax of HTML5 should be mostly familiar

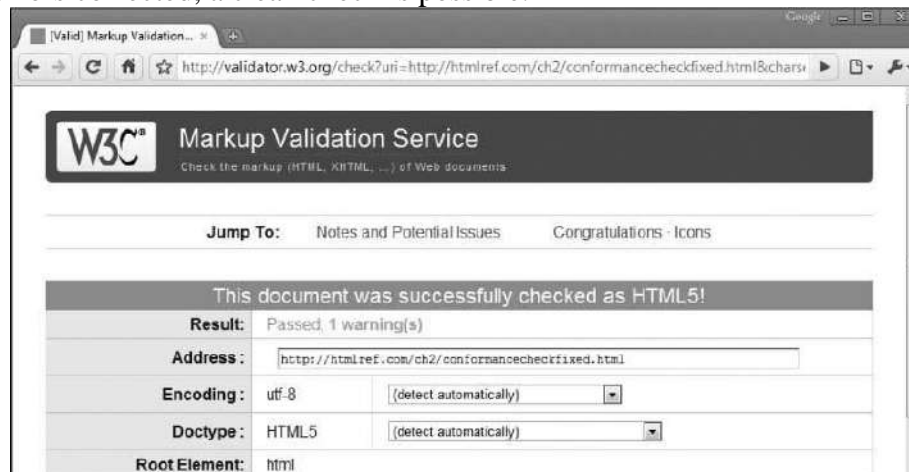
```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Hello HTML5 World</title>
</head>
<body>
<h1>Hello HTML5</h1>
<p>Welcome to the future of markup!</p>
</body>
</html>
```

As indicated by its atypical **<!DOCTYPE>** statement, HTML5 is not defined as an SGML or XML application. Because of the non-SGML/XML basis for HTML, there is no concept of validation in HTML5; instead, an HTML5 document is checked for conformance to the specification, which provides the same practical value as validation.

When checked with an HTML5 conformance checker, such as the W3C Markup Validation Service used in this chapter (available at <http://validator.w3.org>), you see the expected result:



Later, with errors corrected, a clean check is possible:



Loose Syntax Returns

An interesting aspect of HTML5 is the degree of syntax variability that it allows. Unlike its stricter markup cousin, XHTML, the traditional looseness of HTML is allowed. To demonstrate, in the following example, quotes are not always employed, major elements like **html**, **head**, and **body** are simply not included, the inference of close of tags like `</p>` and `` is allowed, case is used variably, and even XML-style self-identifying close syntax is used at will:

To ensure that you conform to the HTML5 specification, you should be concerned primarily about the following:

- **Make sure to nest elements, not cross them; so**

`<i>`is in error as tags cross`</i>`

whereas

`<i>`is not since tags nest`</i>`.

- **Quote attribute values when they are not ordinal values, particularly if they contain special characters, particularly spaces; so**

`<p id=p1>`Fine with no quotes`</p>` because it is a simple attribute value, whereas

`<p title=trouble here with no quotes>`Not ok without quotes`</p>` is clearly messed up.

- **Understand and follow the content model. Just because one browser may let you use a list item anywhere you like,**

``I should be in a list!``

it isn't correct. Elements must respect their content model, so the example should read instead as

``All is well I am in a list!``

- **Do not use invented tags unless they are included via some other markup language:**

`<p>`I `<danger>`shouldn't`</danger>` conform unless I am defined in

another specification and use a name space`</p>`

- **Encode special characters, particularly those used in tags (< >), either as an entity** of a named form, such as `<`, or as a numeric value, such as `<`;

XHTML5

If you want to pursue an “XMLish” approach to your document, HTML5 allows it. Consider, for example, a strict XHTML example that is now HTML5:

XHTML5 usage clearly indicates that an HTML5 document written to XML syntax must be served with the MIME type `application/xhtml+xml` or `application/xml`. Unfortunately, although HTML5 supports XML, the real value of XHTML—the true strictness of XML—has not been realized, at least so far, because of a lack of browser support. You can write XMLish markup and serve it as `text/html` but it won't provide the benefit of strict syntax conformance. In short, HTML5 certainly allows you to try to continue applying the intent of XHTML in the hopes that someday it becomes viable.

HTML5: Embracing the Reality of Web Markup

The harsh reality is that, indeed, valid markup is more the exception than the rule online. Numerous surveys have shown that in the grand scheme of things, few Web sites validate. For example

Study	Date	Passed validation	Total validated	Percentage
Parnas	Dec. 2001	14,563	2,034,788	0.71%
Saarsou	Jun. 2006	25,890	1,002,350	2.58%
MAMA	Jan. 2008	145,009	3,509,180	4.13%

Fig 5-1: Validation pass rate studies

The permissive nature of browsers is required for browsers to fix markup mistakes. HTML5 directly acknowledges this situation and aims to define how browsers should parse both well formed and malformed markup, as indicated by this brief excerpt from the specification:

This specification defines the parsing rules for HTML documents, whether they are syntactically correct or not. Certain points in the parsing algorithm are said to be *parse errors*.

Presentational Markup Removed and Redefined

HTML5 removes a number of elements and attributes. Many of the elements are removed because they are more presentational than semantic.

Looking at Table 2-1, you might notice that some elements that apparently should be eliminated somehow live on. For example, `<small>` continues to be allowed, but `<big>` is obsolete. The idea here is to preserve elements but shift meaning.

Removed HTML Element	CSS Equivalent
<code><basefont></code>	<code>body {font-family: family; font-size: size;}</code>
<code><big></code>	<code>font-size: larger</code>
<code><center></code>	<code>text-align: center</code> or <code>margin: auto</code> depending on context
<code></code>	<code>font-family, font-size, or font</code>
<code><s></code> , <code><strike></code>	<code>text-decoration: strike</code>
<code><tt></code>	<code>font-family: monospace</code>
<code><u></code>	<code>text-decoration: underline</code>

TABLE 2-1 HTML 4 Elements Removed from HTML5

Out with the Old Elements

A few elements are removed from the HTML5 specification simply because they are archaic, misunderstood, have usability concerns, or have a function that is equivalent to the function of other elements. Table 2-4 summarizes some of the elements that have been removed from the HTML5 specification.

HTML Element	New Meaning in HTML5
<code></code>	Represents an inline run of text that is different stylistically from normal text, typically by being bold, but conveys no other meaning of importance.
<code><dd></code>	Used with HTML5's new details and figure elements to define the contained text. Was also used with a dialog element which was later removed from the HTML5 specification.
<code><dt></code>	Used with HTML5's new details and figure element to summarize the details. Was also used with a dialog element which was later removed from the HTML5 specification.
<code><hr></code>	Represents a thematic break rather than a horizontal rule, though that is the likely representation.
<code><i></code>	Represents an inline run of text in an alternative voice or tone that is supposed to be different from standard text but that is generally presented in italic type.
<code><menu></code>	Redefined to represent user interface menus, including context menus.
<code><small></code>	Represents small print, as in comments or legal fine print.
<code></code>	Represents importance rather than strong emphasis.

TABLE 2-2 HTML 4 Elements Redefined in HTML5

Attribute Removed	Elements Effected	CSS Equivalent
align	caption, col, colgroup, div, iframe, h1, h2, h3, h4, h5, h6, hr, img, input, legend, object, p, table, tbody, td, tfoot, th, thead, tr	text-align or in some block element cases float
alink	body	body a:active {color: color-value;}
background	body	background-image or background
bgcolor	body, table, td, th, tr	background-color
border	img, object, table	border-width and/or border
cellpadding	table	padding
cellspacing	table	margin
char	col, colgroup, table, tbody, td, tfoot, th, thead, tr	N/A
charoff	col, colgroup, table, tbody, td, tfoot, th, thead, tr	N/A
clear	br	clear
compact	dl, menu, ol, ul	margin properties
frame	table	border properties
frameborder	iframe	border properties
height	td, th	height
hspace	img, object	margin properties
link	body	body a:link {color: color-value;}
marginheight	iframe	margin properties
marginwidth	iframe	margin properties
noshade	hr	border-style or border
nowrap	td, th	overflow
rules	table	border properties
scrolling	iframe	overflow
size	hr	width
text	body	body {color: color-value;}
type	li, ol, ul	list-style-type and list-style
valign	col, colgroup, tbody, td, tfoot, th, thead	vertical-align
vlink	body	body a:visited {color: color-value;}
width	col, colgroup, hr, pre, table, td, th	width

TABLE 2-3 HTML 4 Attributes Removed in HTML5

Removed Element	Reasoning	Alternatives
acronym	Misunderstood by many Web developers.	Use the abbr element.
applet	Obsolete syntax for Java applets.	Use the object element.
dir	Rarely used, and provides similar functionality to unordered lists.	Use the ul element.
frame	Usability concerns.	Use fixed-position elements with CSS and/or object elements with sourced documents.
frameset	Usability concerns.	Use fixed-position elements with CSS and/or object elements with sourced documents.
isindex	Archaic and can be simulated with typical form elements.	Use the input element to create text field and button and back up with appropriate server-side script.
noframes	Since frames are no longer supported, this contingency element is no longer required.	N/A

TABLE 2-4 Elements Removed by HTML5

In with the New Elements

For most Web page authors, the inclusion of new elements is the most interesting aspect of HTML5. many browsers are implementing a few of the more interesting ones, such as **audio** and **video**, and others can easily be simulated.

Sample of New Attributes for HTML5

One quite important aspect of HTML5 is the introduction of new attributes. There are quite a few attributes that are global and thus found on all elements. Table 2-6 provides a brief overview of these attributes.

New Element	Description
article	Encloses a subset of a document that forms an independent part of a document, such as a blog post, article, or self-continued unit of information.
aside	Encloses content that is tangentially related to the other content in an enclosing element such as section .
audio	Specifies sound to be used in a Web page.
canvas	Defines a region to be used for bitmap drawing using JavaScript.
command	Located within a menu element, defines a command that a user may invoke.
datalist	Indicates the data items that may be used as quick choices in an input element of type="list" .
details	Defines additional content that can be shown on demand.
figure	Defines a group of content that should be used as a figure and may be labeled by a legend element.
footer	Represents the footer of a section or the document and likely contains supplementary information about the related content.
header	Represents the header of a section or the document and contains a label or other heading information for the related content.

hgroup	Groups heading elements (h1–h6) for sectioning or subheading purposes.
mark	Indicates marked text and should be used in a similar fashion to show how a highlighter is used on printed text.
meter	Represents a scalar measurement in a known range similar to what may be represented by a gauge.
nav	Encloses a group of links to serve as document or site navigation.
output	Defines a region that will be used to hold output from some calculation or form activity.
progress	Indicates the progress of a task toward completion, such as displayed in a progress meter or loading bar.
rp	Defines parentheses around ruby text defined by an rt element.
rt	Defines text used as annotations or pronunciation guides. This element will be enclosed within a ruby element.
ruby	This is the primary element and may include rt and rp elements. A ruby element serves as a reading or pronunciation guide. It is commonly used in Asian languages, such as in Japanese to present information about Kanji characters.
section	Defines a generic section of a document and may contain its own header and footer .
source	Represents media resources for use by audio and video elements.
source	Represents media resources for use by audio and video elements.
time	Encloses content that represents a date and/or time.
video	Includes a video (and potentially associated controls) in a Web page.

TABLE 2-5 Elements Added by HTML5

New Attribute	Description
accesskey	Defines the accelerator key to be used for keyboard access to an element.
contenteditable	When set to true , the browser should allow the user to edit the content of the element. Does not specify how the changed content is saved.
contextmenu	Defines the DOM id of the menu element to serve as a context menu for the element the attribute is defined on.
data-X	Specifies user-defined metadata that may be put on tags without concern of collision with current or future attributes. Use of this type of attribute avoids the common method of creating custom attributes or overloading the class attribute.
draggable	When specified, should allow the element and its content to be dragged.
hidden	Under HTML5, all elements may have hidden attribute which when placed indicates the element is not relevant and should not be rendered. This attribute is similar to the idea of using the CSS display property set to a value of none .
itemid	Sets a global identifier for a microdata item. This is an optional attribute, but if it is used, it must be placed in an element that sets both the itemscope and itemtype attributes. The value must be in the form of a URL.
itemprop	Adds a name/value pair to an item of microdata. Any child of a tag with an itemscope attribute can have an itemprop attribute set in order to add a property to that item.

itemref	Specifies a list of space-separated elements to traverse in order to find additional name/value pairs for a microdata item. By default, an item only searches the children of the element that contains the itemscope attribute. However, sometimes it does not make sense to have a single parent item if the data is intermingled. In this case, the itemref attribute can be set to indicate additional elements to search. The attribute is optional, but if it is used, it must be placed in an element that sets the itemscope attribute.
itemscope	Sets an element as an item of microdata (see "Microdata" later in the chapter).
itemtype	Defines a global type for a microdata item. This is an optional attribute, but if it is used, it must be placed in an element that sets the itemscope attribute. The value must be in the form of a URL.
spellcheck	Enables the spell checking of an element. The need for this attribute globally may not be clear until you consider that all elements may be editable at page view time with the contenteditable attribute.
tabindex	Defines the element-traversal order when the keyboard is used for navigation.

TABLE 2-6 Key Attributes Added by HTML5

HTML5 Document Structure Changes

HTML5 document structure seems pretty much the same as in HTML 4 save a slightly different **<!DOCTYPE>** statement.

HTML5 documents may contain a **header** element, which is used to set the header section of a document and thus often contains the standard **h1** to **h6** heading elements:

```
<header>
<h1>Welcome to the Future World of HTML5.</h1>
<h2>Don't be scared it isn't that hard!</h2>
</header>
```

Similarly, a **footer** element is provided for document authors to define the footer content of a document, which often contains navigation, legal, and contact information:

```
<footer>
<p>Content of this example is not under copyright</p>
</footer>
```

The HTML5 structural element with the most possible uses is the **section** element. A particular **<section>** tag can be used to group arbitrary content together and may contain further **<section>** tags to create the idea of subsections.

```
<section>
<h1>Chapter 2</h1>
<p>New HTML5 elements.</p>
<section>
<h2>HTML5's section Element</h2>
<p>These elements are useful to create outlines.</p>
<section>
<h3>Nest Away!</h3>
<p>Nest your sections but as you nest you might want to indent.</p>
</section>
</section>
<p>Ok that's enough of that.</p>
</section>
```

It may not be obvious but a **section** element may contain **header** and **footer** elements of its own:

HTML5 uses headings and newly introduced elements like the **section** element for outlining purposes.

join the subhead to the headline with an **hgroup** element like so:

```
<header>
<hgroup>
  <h1>Welcome to the Future World of HTML5</h1>
  <h2>Don't be scared it isn't that hard!</h2>
</hgroup>
</header>
```

- 1. Welcome to the Future World of HTML5
 - 1. Don't be scared it isn't that hard!
 - 2. Chapter 2
 - 1. Introduction to HTML 5
 - 2. New Structural Elements
 - 1. header Element
 - 2. footer Element
 - 3. section Element
 - 3. New Form Elements
 - 1. input type=date
 - 3. Chapter 3

No **hgroup**
elements used

- 1. Welcome to the Future World of HTML 5
 - 1. Chapter 2
 - 1. New Structural Elements
 - 1. header Element
 - 2. footer Element
 - 3. section Element
 - 2. New Form Elements
 - 1. input type=date
 - 2. Chapter 3

hgroup
elements used

Beyond sectioning, HTML5 introduces a number of other structural elements. For example, the **article** element is used to define a discrete unit of content such as a blog post, comment, article, and so on.

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>HTML5 article example</title>
</head>
<body>

<header>
  <hgroup>
    <h1>Welcome to the Future World of HTML5 Blog</h1>
    <h2>Don't be scared it isn't that hard!</h2>
  </hgroup>
</header>
<section id="articleList">
  <h2>Latest Posts</h2>

  <article id="article3">
    <h2>HTML5 Here Today!</h2>
    <p>Article content here...</p>
  </article>
```

The idea of defining these discrete content units specially is that you might wish to extract them automatically, so again, having defined elements as opposed to some ad hoc use of **class** names on **<div>** tags is preferred.

HTML5 also introduces an **aside** element, which may be used within content to represent material that is tangential or, as the element name suggests, an aside:

```
<p>Here we explore the various HTML5 elements. I would write
some real content here but you are busy reading the book anyway.
</p>
<aside>
  <h2>Pointless Aside</h2>
  <p>Oh by the way did you know that the author lives in San Diego?
  It is completely irrelevant to the discussion but he seems
  to like the weather there as opposed to rainy New Zealand.</p>
</aside>

<p>So as we continue to discuss the various HTML5 elements we must
remember to stay focused as there is much to learn.
</p>
```

Adding Semantics

Many of the elements that HTML5 adds that can be used right away are semantic in nature. In this sense, HTML5 continues the appropriate goal of separating structure from style.

Marking Text

The new HTML5 element **mark** was introduced for highlighting content similarly to how a highlighter pen might be used on important text in a book. The following example wraps a few important words:

```
<p>Here comes <mark>marked text</mark> was it obvious?</p>
```

Unfortunately, you won't necessarily see anything with such an example:

Here comes marked text was it obvious?

You would need to apply a style. Here, inline styles are used just to show the idea:

```
<p>The new HTML5 specification is in the works. While <mark
style="background-color: red;">many features are not currently
implemented or even well defined</mark> yet, <mark
style="background-color: green;">progress is being made</mark>.
Stay tuned to see more new HTML elements added to your Web documents
in the years to come.</p>
```

The new HTML5 specification is in the works. While many features are not currently implemented or even well defined yet, progress is being made. Stay tuned to see more new HTML elements added to your Web documents in the years to come.

Indicating Dates and Time

Another semantic inline element, **time**, was introduced by HTML5 to indicate content that is a date, time, or both. For example,

```
<p>Today it is <time>2009-07-08</time> which is an interesting date.</p>
```

as well as

```
<p>An interesting date/time for SciFi buffs is <time>1999-09-13T09:15:00
</time>!</p>
```

would both be valid. The element should contain a date/time value that is in the format YYYY-MM-DDThh:mm:ssTZD, where the letters correspond to years, months, days, hours, minutes, and seconds, T is the actual letter 'T,' and ZD represents a time zone designator of either Z or a value like +hh:mm to indicate a time zone offset.

If you try something like

```
<p>Right now it is <time>6:15</time>.</p>
```

it may be meaningful to you but it does not conform to HTML5.

The following example is meaningful because it provides both a readable form and a machine-understood value:

```
<p>My first son was born on <time datetime="2006-01-13">Friday the 13th</time> so it is my new lucky day.</p>
```

Inserting Figures

It is often necessary to include images, graphs, compound objects that contain text and images, and so on in our Web documents, all of which usually are called figures. HTML5 reintroduces the idea with the more appropriately named **figure** element. A simple example illustrates this new element's usage:

```
<figure id="fig1">
  <dd>
    
    <p>This mighty &lt;figure&gt; tag has returned from HTML 3 to haunt your
      dreams.</p>
  </dd>
  <dt>Figure Ex-1</dt>
</figure>
```

Acting as a semantic element, **figure** simply groups items within an enclosed `<dd>` tag, though it may associate them with a caption defined by a `<dt>` tag as shown in the example.

Specifying Navigation

One new HTML5 element that is long overdue is the **nav** element. The purpose of this element is to encapsulate a group of links that serves as a collection of offsite links, document navigation, or site navigation:

```
<nav>
  <h2>Offsite Links</h2>
  <a href="http://www.w3c.org">W3C</a><br>
  <a href="http://www.htmlref.com">Book site</a><br>
  <a href="http://www.pint.com">Author's Firm</a><br>
</nav>
```

The semantics defined by HTML5 for a `<nav>` tag eliminate this confusion. Interestingly, there is no requirement to avoid using `` and `` tags within navigation, so if you are a CSS aficionado who is comfortable with that approach, it is fine to use markup like this:

```
<nav id="mainNav">
<ul>
  <li><a href="about.html">About</a></li>
  <li><a href="services.html">Services</a></li>
  <li><a href="contact.html">Contact</a></li>
  <li><a href="index.html">Home</a></li>
</ul>
</nav>
```

HTML5's Open Media Effort

<video>

To insert video, use a **<video>** tag and set its **src** attribute to a local or remote URL containing a playable movie. You should also display **playback** controls by including the **controls** attribute, as well as set the dimensions of the movie to its natural size. This simple demo shows the use of the new element:

```
<video src="http://htmlref.com/ch2/html_5.mp4"
       width="640" height="360" controls>
<strong>HTML5 video element not supported</strong>
</video>
```



To address the media support problem, you need to add in alternative formats to use by including a number of **<source>** tags:

```
<video width="640" height="360" controls poster="loading.png">
<source src="html_5.mp4" type="video/mp4">
<source src="html_5.ogv" type="video/ogg">
<strong>HTML5 video element not supported</strong>
</video>
```

Also note in the preceding snippet the use of the **poster** attribute, which is set to display an image in place of the linked object in case it takes a few moments to load. Other **video** element specific attributes like **autobuffer** can be used to advise the browser to download media content in the background to improve playback, and **autoplay**, which when set, will start the media as soon as it can.

<audio>

HTML5's **audio** element is quite similar to the **video** element. The element should support common sound formats such as WAV files:

```
<audio src="http://htmlref.com/ch2/music.wav"></audio>
```

If you want to allow the user to control sound play, unless you have utilized JavaScript to control this, you may opt to show controls with the same named attribute. Depending on the browser, these controls may look quite different, as shown next.



As with the **video** element, you also have **autobuffer** and **autoplay** attributes for the **audio** element. Unfortunately, just like **video**, there are also **audio** format support issues, so you may want to specify different formats using **<source>** tags:

```
<audio controls autobuffer autoplay>
<source src="http://htmlref.com/ch2/music.ogg" type="audio/ogg">
<source src="http://htmlref.com/ch2/music.wav" type="audio/wav">
</audio>
```

Media Considerations

An interesting concern about “open” media formats is whether or not they really are open. As the HTML5 specification emerges, fissures are already forming in terms of how these elements are implemented, what codecs will be supported by what browser vendors, and whether HTML5 will require a particular codec to be supported by all HTML5-compliant browsers.

The following adds in a fallback within the previous video example for Flash:

```
<video width="640" height="360" controls poster="loading.png">
<source src="http://htmlref.com/ch2/html_5.mp4" type="video/mp4">
<source src="http://htmlref.com/ch2/html_5.ogv" type="video/ogg">

<object data="html_5.swf" type="application/x-shockwave-flash"
width="640" height="360" id="player">
<param name="movie" value="html_5.swf"/>
<strong>Error: No video support at all</strong>
</object>
</video>
```

Client-Side Graphics with <canvas>

The **canvas** element is used to render simple graphics such as line art, graphs, and other custom graphical elements on the client side. From a markup point of view, there is little that you can do with a **<canvas>** tag. You simply put the element in the page, name it with an **id** attribute, and define its dimensions with **height** and **width** attributes:

```
<canvas id="canvas" width="300" height="300">
<strong>Canvas Supporting Browser Required</strong>
</canvas>
```

After you place a **<canvas>** tag in a document, your next step is to use JavaScript to access and draw on the element. For example, the following fetches the object by its **id** value and creates a two-dimensional drawing context:

```
var canvas = document.getElementById("canvas");
var context = canvas.getContext("2d");
```

Drawing and Styling Lines and Shapes

HTML5 defines a complete API for drawing on a **canvas** element, which is composed of many individual sub-APIs for common tasks. For example, to do some more complex shapes, the path API must be used. The path API stores a collection of subpaths formed by various shape functions and connects the subpaths via a fill() or stroke() call. To begin a path, context.beginPath() is called to reset the path collection. Then, any variety of shape calls can occur to add a subpath to the collection. Once all subpaths are properly added, context.closePath() can optionally be called to close the loop. Then fill() or stroke() will also display the path as a newly created shape. This simple example draws a V shape using lineTo():

```
context.beginPath();  
context.lineTo(20,100);  
context.lineTo(120,300);  
context.lineTo(220,100);  
context.stroke();
```



Now, if you were to add context.closePath() before context.stroke(), the V shape would turn into a triangle, because closePath() would connect the last point and the first point.

Also, by calling fill() instead of stroke(), the triangle will be filled in with whatever the fill color is, or black if none is specified. style the drawing, you can specify the fillStyle and strokeStyle and maybe even define the width of the line using lineWidth, as shown in this example:

```
context.strokeStyle = "blue";  
context.fillStyle = "red";  
  
context.lineWidth = 10;  
context.beginPath();  
context.lineTo(200,10);  
context.lineTo(200,50);  
context.lineTo(380,10);  
context.closePath();  
context.stroke();  
context.fill();
```



HTML5 Form Changes

However, most of the Web Forms specification has been incorporated into HTML5 and more and more of its features are now being implemented in browsers.

New Form Field Types

Traditionally, the HTML **input** element is used to define most form fields. The particular type of form field of interest is defined with the **type** attribute, which is set to **text**, **password**, **hidden**, **checkbox**, **radio**, **submit**, **reset**, **image**, or **button**. HTML5 adds quite a number of other values, which we will briefly explore here.

First, setting the **type** equal to **color** should create a color picker:

```
<p><label>color:<input type="color" name="favColor"></label></p>
```

A variety of date controls can now be directly created by setting the **type** attribute to **date**, **datetime**, **datetime-local**, **month**, **week**, or **time**. Several of these controls are demonstrated here:

```

<p><label>date:
  <input type="date" name="date">
</label></p>

<p><label>datetime:
  <input type="datetime" name="datetime">
</label></p>

<p><label>datetime-local:
  <input type="datetime-local" name="datetime2">
</label></p>

<p><label>time:
  <input type="time" name="time">
</label></p>

<p><label>month:
  <input type="month" name="month">
</label></p>

<p><label>week:
  <input type="week" name="week">
</label></p>

```

Pick a Date:

Pick a Date and Time: 2020-02-04 01:00

Pick a Date and Time (local):

month: 2009-10

week: 2009-W43

time: 23:52

Setting **type** to **number** gives you a numeric spin box in conforming browsers:

```
<p><label>number:<input type="number" name="number"></label></p>
```

For example, the **max** attribute can be set to limit the maximum value, **min** to limit the smallest value, and even **step** to indicate how values may be modified. For example,

```
<input type="range" name="range" max="100" min="1" step="5">
```

A similar form of control can be created using a **range** control:

```
<input type="range" name="range" max="100" min="1" step="5">
```

This control presents itself as a slider, which so far has a varied appearance in browsers:

Like the number picker, the **min**, **max**, and **step** attributes all can be set to limit values:


```
<p><label>range (1-100 step 5):  
<input type="range" name="range" max="100" min="1" step="5">  
</label></p>
```

It is also possible to further define semantic restrictions by setting an `<input>` tag's **type** attribute to **tel**, **email**, or **url**:

```
<p><label>Telephone Number: <input type="tel" name="telno"></label></p>  
<p><label>Email: <input type="email" name="email"></label></p>  
<p><label>URL: <input type="url" name="url"></label></p>
```

Semantic Field Types



Validating Data Entry

It is also possible to force the user to enter data, without resorting to JavaScript, in an HTML5-compliant browser by setting the **required** attribute for a form control:

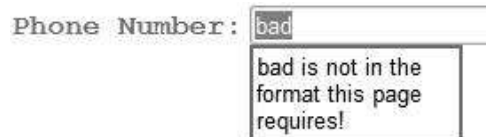
```
<input type="text" name="firstname" id="firstname" required>
```

A browser may then set an error style on the field and present a message if there is a problem:



The **pattern** attribute also can be employed to force the entered data to conform to a supplied regular expression:

```
<label for="phonenumber" class="required">Phone Number:</label>  
<input type="text" name="phonenumber" id="phonenumber" required  
pattern="^\(\d{3}\) \d{3}-\d{4}$">
```



If a **title** is specified when patterns are applied, the browser may display this advisory information:

```
<label for="phonenumber" class="required">Phone Number:</label>  
<input type="text" name="phonenumber" id="phonenumber" required  
pattern="^\(\d{3}\) \d{3}-\d{4}$"  
title="Phone number of form (xxx) xxx-xxxx required">
```



However, in some cases, you can not only apply a **pattern** but also employ the appropriate semantic type value like **email**, though it isn't clear if these elements will apply their own implied validation pattern matches simply by setting them as **required**:

Autocomplete Lists

Under HTML5, the **input** element's **list** attribute is used to set the DOM **id** of a **datalist** element used to provide a predefined list of options suggested to the user for entry:

```
<p><label>Favorite Dog: <input type="text" list="dogs"></label></p>
<datalist id="dogs">
  <option>Angus</option>
  <option>Tucker</option>
  <option>Cisco</option>
  <option>Sabrina</option>
</datalist>
```

Favorite Dog:

- Angus
- Tucker
- Cisco
- Sabrina

Miscellaneous Usability Improvements

Commonly, Web page authors use the **value** attribute to populate some text in a form field:

```
<input type="text" name="firstname" id="firstname" value="Thomas">
```

Quite often, people put placeholder or advisory text here, like so:

```
<input type="text" name="middlename" id="middlename"
value="Enter your middle 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>
```

Emerging Elements and Attributes to Support Web Applications

HTML5 tends to encourage and that some of them may be changed or removed. As of yet, no native

implementation of these elements exists, so we simulated their possible renderings using a JavaScript library.

Menu Element Repurposed

One element that will be implemented in browsers but might not perform the actions defined in HTML5 is the **menu** element. Traditionally, this element was supposed to be used to create a simple menu for choices, but most browsers simply rendered it as an unordered list:

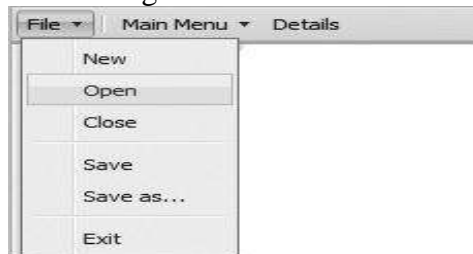
```
<menu type="list" id="oldStyle">
<li>Item 1</li>
<li>Item 2</li>
<li>Item 3</li>
<li>Item 4</li>
</menu>
```

- Item 1
- Item 2
- Item 3
- Item 4

Under HTML5 the **menu** element has been returned to its original purpose. A new attribute, **type**, is introduced that takes a value of **toolbar**, **context**, or **list** (the default). This example sets up a simple File menu for a Web application:

```
<menu type="toolbar" id="fileMenu" label="File">
  <li><a href="javascript:newItem();">New</a></li>
  <li><a href="javascript:openItem();">Open</a></li>
  <li><a href="javascript:closeItem();">Close</a></li>
  <hr>
  <li><a href="javascript:saveItem();">Save</a></li>
  <li><a href="javascript:saveAsItem();">Save as...</a></li>
  <hr>
  <li><a href="javascript:exitApp();">Exit</a></li>
</menu>
```

Using CSS and JavaScript, this menu might render like so:



The global **contextmenu** attribute is used to define an element's context menu, which is generally the menu invoked upon a right-click. The attribute's value should hold a string that references the **id** of a **<menu>** tag found in the DOM. For example,

```
<div contextmenu="simpleMenu">Widget</div>
```

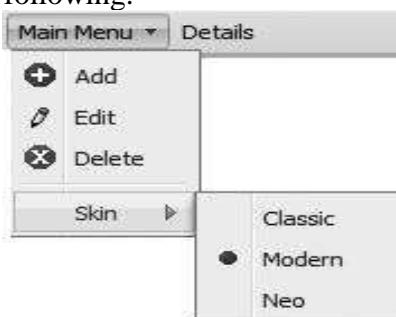
would reference the previously defined **menu** via a right-click. If there is no element found or no value, then the element has no special context menu and the user agent should show its default menu.

command Element

The **menu** element may contain not just links but also other interactive items, including the newly introduced **command** element. This empty element takes a **label** and may have an icon decoration as well. The **command** element has a **type** attribute, which may be set to **command**, **radio**, or **checkbox**, though when **radio** is employed there needs to be a **radiogroup** indication.

```
<menu type="command" label="Main Menu">
  <command type="command" label="Add" icon="add.png">
  <command type="command" label="Edit" icon="edit.png">
  <command type="command" label="Delete" icon="delete.png">
  <hr>
  <menu type="command" label="Skin" id="skinMenu">
    <command type="radio" radiogroup="skin" label="Classic">
    <command type="radio" radiogroup="skin" label="Modern" checked>
    <command type="radio" radiogroup="skin" label="Neo">
  </menu>
  <hr>
  <command type="checkbox" label="Secure Mode">
</menu>
```


Such a menu might look like the following:



Meter and progress Elements

Two fairly similar elements have been introduced in HTML5 to show current status. First, the **meter** element defines a scalar measurement within a known range, similar to what might be represented by a gauge. The following example is a reading of velocity for some fantastically fast space vessel:

```
<p>Warp Drive Output: <meter min="0" max="10" low="3" optimum="7" high="9"
value="9.5" title="Captain she can't take much more of this!"></meter></p>
```

A potential rendering could look like



Slightly different from **meter** is the **progress** element, which defines completion progress for some task. Commonly, this element might represent the percentage from 0 percent to 100 percent of a task, such as loading to be completed:

```
<p>Progress: <progress id="progressBar" max="100.00" value="33.1">
33.1%</progress></p>
```

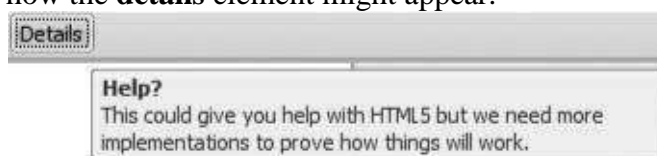


Details Element

The new **details** element is supposed to represent some form of extra details, such as a tooltip or revealed region that may be shown to a user. The **details** element can contain one **dt** element to specify the summary of the details as well as one **dd** element to supply the actual details. The attribute **open** can be set to reveal the details or can be changed dynamically, as shown in this example:

```
<details onclick="this.open='open'">
  <dt>Help?</dt>
  <dd>This could give you help with HTML5 but we need more
    implementations to prove how things will work.</dd>
</details>
```

Here is an example of how the **details** element might appear:



Output Element

The final stop on this speculative tour is the **output** element, which is used to define a region that will be used as output from some calculation or form control. Here I imagine using the calendar picker and having the eventual release date of HTML5 being revealed in an **output** element:

```
<form action="#" method="get" id="testform">
<p><input type="date" id="year">
<p>HTML5 released in the year
<output for="year">&nbsp;</output></p>
</form>
```

Script could certainly be used to perform this action:

08/11/2011 
HTML5 released in the year: 2011

The Uncertain Future of Frames

HTML5 continues to support **<iframe>**; in fact, it not only supports it but extends the tag. The inline frame has plenty of life left if the HTML5 vision comes true because it will be used to include new content and functionality in pages from remote sources and may even be used in intra-document communication. So, the future of frames as far as HTML5 is concerned isn't set.

HTML5 proposes two new attributes for the **iframe** element: **seamless** and **sandbox**. The **seamless** attribute effectively renders the **iframe** as an inline include, which allows the parent document's CSS to affect the contents of the **iframe**:

```
<iframe src="content.html" name="thisframe" width="200"
height="300" seamless>[alternate content]</iframe>
```

Here is the same example using XHTML style syntax:

```
<iframe src="content.htm" name="thisframe" width="200"
height="300" seamless="seamless">[alternate content]</iframe>
```

The **sandbox** attribute “sandboxes” the **iframe**, essentially preventing it from pulling in content from any source other than the **iframe** itself.

These prohibitions can be “turned off” using a number of attributes:

- **allow-same-origin** allows the **iframe** to pull in content from elsewhere in the same domain.
- **allow-forms** permits the submission of forms in the sandboxed **iframe**.
- **allow-scripts** allows the sandboxed **iframe** to run scripts from the same domain.

The order of the attributes does not affect any functionality.

```
<iframe src="content.htm" sandbox="allow-same-origin
allow-forms allow-scripts">
<iframe src="content.htm" sandbox="allow-forms">
```

Under HTML5, the **<iframe>** tag can also be written XHTML style, with a closing slash:

```
<iframe src="content.htm" height="200" width="200"
sandbox="allow-same-origin" />
```

The draggable Attribute and the Drag and Drop API

HTML5 introduces drag and drop natively to the browser. Drag and drop has long been implemented with JavaScript code that was not designed specifically for that purpose. Now the

logic is made much easier and cleaner as the HTML5 specification includes an attribute and events that are intended exclusively for drag and drop.

In order to drag an item, the element must have the **draggable** attribute set to **true**:

```
<div id="dragme" class="box" draggable="true">I am a draggable div</div>
```

Everything else must be configured through JavaScript. There are several new events for drag and drop. These are attached to HTML elements just as any other event using `addEventListener()` or `attachEvent()`.

The following events are attached to the item that will be dragged:

- **dragstart** The drag has begun.
- **drag** The element is being moved.
- **dragend** The drag has completed.

The rest of the events are attached to the drop area:

- **dragenter** The element is dragged into the drop area.
- **dragover** The element is dragged into the drop area. The default behavior here is to cancel the drop, so it is necessary to hook up this event and then return false or call `preventDefault()` to cancel the default behavior.
- **dragleave** The element is dragged out of the drop area.
- **drop** The element is dropped in the drop area.

Here we use JavaScript to hook up some of these events on a draggable box and a drop area:

```
var drag = document.getElementById("dragbox");
drag.addEventListener("dragstart",dragstart,false);
drag.addEventListener("dragend",dragend,false);
var drop = document.getElementById("dropzone");
drop.addEventListener("dragenter",dragenter,false);
drop.addEventListener("dragleave",dragleave,false);
drop.addEventListener("dragover",dragover,false);
drop.addEventListener("drop",drops,false);
```

Content editable Attribute

First introduced by Internet Explorer, the proprietary **contenteditable** attribute is supported by most browsers today. HTML5 standardizes the use of this attribute globally on all elements. The basic sense of the attribute is that if you set it to **true**, the browser should allow you to modify the text directly when you click the element:

```
<p contenteditable="true">This paragraph of text is editable. Click it
and you will see. Of course there is no sense of saving it with code to
transmit the information to the server. This paragraph of text is editable
Click it and you will see. Of course there is no sense of saving it with
code to transmit the information to the server.</p>
```

The browser may or may not present a style change to show you are in “edit mode.”

This paragraph of text is editable. This browser shows an edit mode. Click it and you will see. Of course there is no sense of saving it with code to transmit the information to the server. This paragraph of text is editable. Click it and you will see. Of course there is no sense of saving it with code to transmit the information to the server.

Style change
upon edit

versus

This paragraph of text is editable. Click it and you will see. This browser is not showing an editing style change. Of course there is no sense of saving it with code to transmit the information to the server. This paragraph of text is editable. Click it and you will see. Of course there is no sense of saving it with code to transmit the information to the server.

No style change
upon edit

This paragraph uses some simple script to be editable. Double click the text to begin editing.

It is possible to use JavaScript to enable content editing by changing the corresponding `contentEditable` property for the element. For example, the following changes this property and updates the **class** name to reflect a style change when in edit mode.

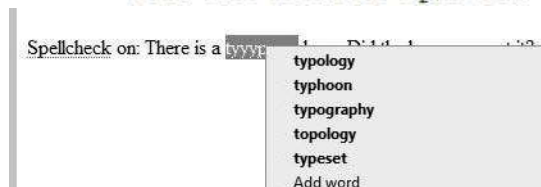
```
<p ondblclick="this.contentEditable=true;this.className='inEdit';"  
onblur="this.contentEditable=false;this.className='';">This paragraph  
uses some simple script to be editable. Double-click the text to  
begin editing.</p>
```

This paragraph uses some simple script to be editable. This is newly inserted text. Double click the text to begin editing.

Spellcheck Attribute

HTML5 defines a **spellcheck** attribute globally for elements. Enabling the spell checking of element content is a matter of setting the **spellcheck** attribute to **true**:

```
<p spellcheck="true">Spellcheck on: There is a tyypoo here.  
Did the browser spot it?</p>
```



Commonly, this attribute is a bit more useful on form fields, given their interactive nature:

```
<label>Text field: (spellcheck on)  
<input type="text" name="textfield" spellcheck="true" value="There is a  
tyypoo here. Did the browser spot it?"></label>
```

Given the application of single-line text fields, it is far more useful to set this attribute on multiline text fields defined by a **<textarea>** tag, like so:

```
<label>Text area: (spellcheck on) <textarea name="comments"  
spellcheck="true">There is a tyypoo here. Did the browser spot it?  
</textarea></label>
```

Internationalization Improvements

While there are not many internationalization-supporting changes in the HTML5 specification, it does make standard the **ruby**, **rp**, and **rt** elements, which were initially supported by the Internet Explorer browsers to associate a reading text with a base text for certain East Asian languages like Japanese. The base text that the annotation is associated with should be enclosed in a **<ruby>** tag; the annotation, enclosed in a **<rt>** tag, will appear as smaller text above the base text, and optionally an **<rp>** tag can be used to wrap content to delimit ruby text for browsers that do not support this formatting:

```
<p>
  <!-- The Kanji for Japanese language with the romanji above it or within
  parens for non ruby aware browsers -->
  <ruby>
    日本語 <rp>(</rp><rt>nihongo</rt><rp>)</rp>
  </ruby>
</p>
```



Questions bank:

1. Why few HTML4 Elements Removed from HTML5? Give the Removed HTML Elements list and also CSS alternatives
2. Write the snippet of code to demonstrate the following semantic inline elements (i) Indicating Dates and Time (ii) Inserting
3. Comment on the internationalization supporting changes in the HTML5 specification.
4. How HTML5 perform the Drawing and Styling Lines and Shapes? Explain with an example to draw few different shapes with fills and styles.
5. What is the use of command Element and its attributes in HTML5. Explain with the help of code snippet.
6. Mention the rules to remember while mappings between markup and script.
7. Explain the role of the following semantic elements of HTML5 with syntax and script segments: i) `<nav>` ii) `<section>` iii) `<aside>`
8. What are the document structure changes in HTML5 with syntax and segments explain the following tags i) `<section>` ii) `<hgroup>` iii) `<article>` iv) `<aside>`
9. Write the division `<div>` based HTML semantic structure element.
10. What do you mean by Loose Syntax Returns in HTML5 briefly explain
11. Mention what are the Out with the Old Elements in HTML5.
12. Write a note on XHTML and HTML5