

Hypertext Markup Language

Giuseppe Della Penna

Università degli Studi di L'Aquila

giuseppe.dellapenna@univaq.it

<http://people.disim.univaq.it/dellapenna>

Document version: 231124

This document is based on the slides of the Web Engineering course, translated into English and reorganized for a better reading experience. It is not a complete textbook or technical manual, and should be used in conjunction with all other teaching materials in the course. Please report any errors or omissions to the author.

This work is licensed under CC BY-NC-SA 4.0. To view a copy of this license, visit <https://creativecommons.org/licenses/by-nc-sa/4.0>

- 1. Introduction to HTML
 - 1.1. Hypertext Markup Language
 - 1.2. Basic notions
 - 1.3. Cross-Browser Compatibility
- 2. Basic HTML document structure
 - 2.1. Beginning a Document
 - 2.2. Basic structure of an HTML document
 - 2.3. Element Classification
 - 2.4. Standard HTML attributes
- 3. Text flow elements
 - 3.1. Paragraphs
 - 3.2. Line breaks
 - 3.3. Sections
 - 3.4. HTML5 structural elements
 - 3.5. Containers
- 4. Text formatting elements
 - 4.1. Semantic Formatting
 - 4.2. Basic Formatting
- 5. Lists and tables
 - 5.1. Lists
 - 5.2. HTML5 menus
 - 5.3. Tables

- 6. Images and embedded objects
 - 6.1. Images
 - 6.2. Image Maps
 - 6.3. Objects
- 7. Meta elements
 - 7.1. Links
 - 7.2. Style Sheets
- 8. Interaction elements
 - 8.1. Links
 - 8.2. Forms
- 9. References

1. Introduction to HTML

1.1. Hypertext Markup Language

The structural part of the web

HyperText Markup Language (HTML) is the most basic building block of the Web, and is used to **define the structure of its content**. HTML allows you to define **hypertexts**, i.e. texts that are connected to each other via links.

Technically, HTML is a **markup language**, in which the textual content is structured using particular delimiters called **tags**, having the generic syntax "`<tag>`". There are opening (`<tag>`) and closing (`</tag>`) tags, just like the opening "(" and closing ")" brackets in common language.

Text delimited by a tag is called an **element**. The tag of each element provides the **semantics** of its content, indicating how it should be interpreted (but not visually represented: CSS style sheets take care of this). The tags are therefore not displayed in browsers, but instruct them how to display the content of the web page.

HTML (like most markup languages) is easily manipulated by machines but also readable by humans.

HTML is also totally **independent from hardware architectures, operating systems and protocols**, and is even **used outside the Web** as an alternative format for structured text similar to that generated by word processors, for example in emails and even for defining software user interfaces.

Evolution

There are numerous versions of HTML, which has evolved following the needs of the web but also of all the other applications that use it:

- 1991 First version (HTML 1.0) defined by Tim Berners-Lee

- 1995 HTML 2.0 (defined by the HTML Working Group)
- 1997 HTML 3.2 (W3C)
- 1999 HTML 4.01 (W3C)
- 2000 XHTML 1.0 (HTML 4.01 with XML syntax)
- 2008 Draft HTML5 (Web Hypertext Application Technology Working Group)
- 2012 HTML5 Living Standard (WHATWG)
- **2014 HTML5 (W3C)**
- 2017 HTML5.2 (W3C)
- ...

The HTML 5 revolution

HTML5 is a **standalone markup language**, not derived from SGML as HTML4.

The syntax of HTML5 is compatible with the one of HTML4 and XHTML1, but does not support some features of HTML4 syntax coming from SGML, such as *processing instructions*.

As W3C (<http://www.w3.org/TR/html5-diff>) says: "*The HTML[5] specification reflects an effort, started in 2004, to study contemporary HTML implementations and deployed content. The draft defines **a single language called HTML which can be written in HTML syntax and in XML syntax**, defines detailed processing models to foster interoperable implementations, **improves markup for documents**, **introduces markup and APIs for emerging idioms**, such as Web applications [...]*"

HTML5 became a **W3C Recommendation** (therefore a final specification) on **October 28, 2014**, but *it is still an evolving standard*. It is therefore always necessary to ensure that an HTML5 feature is adequately supported by current browsers before including it in your projects.

In this document, in the description of the language, the most important changes introduced by HTML5 will be highlighted.

1.2. Basic notions

Text encoding

HTML pages, like the entire Web, are made up mostly of text, so it is important to understand how this can be represented on a machine.

A **character set** (*charset*) defines the set of characters (letters, numbers, punctuation, symbols,...) necessary for a particular purpose (not necessarily related to digital transmission and preservation).

An **encoded character set** is a set of characters that have each been assigned a unique number. The elements of an encoded character set are also known as **code points**. A code point, therefore, represents the position of a character in the encoded character set: for example, the code point for the letter "á" in the Unicode encoded character set is 225.

Character encoding finally defines the way in which the encoded character set (or rather its numerical code points) will be mapped onto bytes to be saved on digital media and transmitted over the network.

Many historical character encoding standards (for example ISO 8859, which is still widely used) use a single byte for each code point which simply represents the position of the character in the set. For example, the "A" in the ISO 8859-1 code set is the 65th character, and is therefore encoded as a byte with the value of 65.

When communicating textual information it is therefore necessary to specify the set in use and the corresponding encoding, so that these characters can be represented reliably.

Unicode

The **Unicode Consortium** has defined a large character set that includes all those needed for any writing system in the world.

The Unicode standard is fundamental to the architecture of the Web and operating systems, and is gradually replacing specific character sets created in the past by various organizations, such as ISO.

The first 65536 code points in the Unicode character set constitute the **Basic Multilingual Plane** (BMP), which includes most of the most commonly used characters. There are also approximately one million additional code points available for **additional characters**, including emoji.

Although the Unicode character set is unique, there are more than one encodings available: **UTF-8**, **UTF-16** and **UTF-32**.

- UTF-8 uses **one byte** to represent characters in the **ASCII** set, **two bytes** for the most common characters in other alphabets, **three bytes** for the rest of the BMP, and **four bytes** for supplementary characters.
- UTF-16 uses 2 bytes for any character in the BMP and 4 bytes for additional characters.
- UTF-32 uses 4 bytes for all characters.

So, although the code point for the letter "á" in the Unicode character set is always 225, in UTF-8 it is represented by two bytes.

In HTML, and on the Internet in general, **Unicode with UTF-8 encoding is currently considered the standard**.

Incredibly, even the malicious use of Unicode character encodings can exploit some vulnerabilities that allow, for example, the theft of information.

UTF-7, originally defined to encode BMP only, allowed alternative encodings of ASCII characters such as "<" and ">". Older versions of Internet Explorer could be tricked into interpreting a page as UTF-7, and in this case the sequences +ADw- and +AD4-, which most validators treat as plain text, were transformed into "<" and ">" allowing XSS attacks.

Since many characters from different languages can visually resemble each other, it is possible to trick a user into browsing a site whose address is visually similar to that of a secure site, even if the characters it contains are actually different, and will therefore lead to a malicious site (**homography** attacks), for example using the Cyrillic character "a" (Unicode 0430) instead of the ASCII character (Unicode 0041)

Resource identifiers, URI and URL

To uniquely identify resources, even outside the network, "**Uniform Resource Identifiers**" (URIs) are commonly used:

```
<schema>://<authority>/<path>?<query>#<fragment>
```

- The mandatory *schema* provides information on the format and meaning of the rest of the URI
- The *authority*, if present, indicates the organization to which the name defined by the rest of the URI belongs
- The *path*, mandatory, provides the first part of the unique name (possibly with respect to the authority) of the resource, represented in a hierarchical manner
- The *query*, if present, provides an additional part of the name, this time in the form of non-hierarchical data
- The *fragment*, if present, allows one to point to a sub-resource within the main resource

A "**Uniform Resource Locator**" (URL) is a specific type of URI that not only identifies the resource, but indicates where it is located and/or how to access it.

```
<protocol>://<host>:<port>/<path>?<query>#<fragment>
```

In a URL, the *scheme* identifies the **protocol** for accessing the resource (http, ftp,...)

The *authority* consists of the FQDN of the **host** hosting the resource (or its IP address) possibly followed by the *port* number that identifies the logical service to connect to on the host (if omitted, it has a protocol-dependent default)

The **path** is a sequence of strings separated by slashes (/) which, as in a filesystem, identifies a resource through the logical path necessary to reach it (in the case of the filesystem, how to reach a file by crossing the folder hierarchy).

Resource representation

The resources exchanged over a network must be adequately **represented** in order to be understandable by the devices that are communicating.

For example, when you transmit an image you are not actually transmitting the image, but a

representation of it, encoded for example using the JPEG format.

For communication to be effective, the two parties must agree not only on the protocol, but also on the formats to be used to exchange resources.

On the Internet there are a large number of **standard formats** for the representation of numerous classes of resources, such as HTML for hypertexts, JPEG, PNG, GIF, BMP, etc. for images, various derivatives of MPEG for audio and video, etc.

These types are indicated via so-called **media types** (also known as **MIME** types), which have the generic form "*type/subtype*", for example text/html, image/jpeg, audio/mpeg.

1.3. Cross-Browser Compatibility

Standards and Quirks Mode

Browsers support two rendering modes: *Standards mode* and *Quirks mode*.

- Standards mode works as close as possible to the W3C specifications, so it is (almost) browser independent.
- Quirks mode follows the formatting rules of the specific browser, with its limitations and extensions.

Quirks mode exists only for compatibility with old sites, which were developed with very *browser-dependent* code. Today, it is *necessary* to develop new sites in Standards mode.

By default browsers use Quirks mode. To enter Standards mode, a specific declaration must be placed at the beginning of the document.

In particular, in the HTML5 standards mode:

Documents using HTML5 with **HTML syntax** must be served with media type **text/html** and must begin with the (pseudo) DOCTYPE declaration `<!doctype html>`.

Documents using HTML5 with **XML syntax** must be served with *media types* such as **application/xml**, **application/xhtml+xml**, **text/xml**, must contain the XML declaration, and declare the namespace **http://www.w3.org/1999/xhtml** in the `<html>` element. DOCTYPE declarations are not required if the document is served with the correct media type.

2. Basic HTML document structure

2.1. Beginning a Document

XHTML (HTML4)

```
<!--
XHTML Strict
(Content-type: application/xhtml+xml )
-->

<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD

<html xmlns="http://www.w3.org/1999/xhtml" >
  <head>
    <title>...</title>
  </head>
  <body> ... </body>
</html>
```

```
<!--
XHTML Transitional
(Content-type: application/xhtml+xml )
-->

<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD

<html xmlns=" http://www.w3.org/1999/xhtml" >
  <head>
    <title>...</title>
  </head>
  <body> ... </body>
</html>
```

HTML5

```

<!--
HTML5 with HTML syntax
(Content-type: text/html)
-->

<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>...</title>
  </head>
  <body> ... </body>
</html>

<!--
(also <!DOCTYPE html SYSTEM "about:legacy-compat">)
-->

```

```

<!--
HTML5 with XML syntax
(Content-type: application/xhtml+xml )
-->

<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml " >
  <head>
    <title>...</title>
  </head>
  <body> ... </body>
</html>

```

2.2. Basic structure of an HTML document

`<html>` : root element

Content: `<head>` , `<body>` (both mandatory)

Attributes: lang, dir

This element opens the HTML document.

`<head>` : heading

Content: `<title>` (mandatory), `<base>` , `<script>` , `<meta>` , `<link>` , `<style>`

Attributes: lang, dir

Contains information about the document that usually do not produce any output but affect the logic and the presentation of the document itself.

`<body>` : body

Content: block, `<script>`, `<ins>`, ``

Attributes: standard HTML

Contains the HTML document itself.

2.3. Element Classification

Inline content is represented by text and by the following elements: `<tt>`, `<i>`, ``, `<big>`, `<small>`, ``, ``, `<dfn>`, `<code>`, `<samp>`, `<kbd>`, `<var>`, `<cite>`, `<abbr>`, `<acronym>`, `<a>`, ``, `<object>`, `
`, `<script>`, `<map>`, `<q>`, `<sub>`, `<sup>`, ``, `<bdo>`, `<input>`, `<select>`, `<textarea>`, `<label>`, `<button>`

Elements that generate *block* content are: `<p>`, `<h1>` ... `<h6>`, ``, ``, `<pre>`, `<dl>`, `<div>`, `<noscript>`, `<blockquote>`, `<form>`, `<hr>`, `<table>`, `<fieldset>`, `<address>`

The *flow* content type is given by the union of *inline* and *block*

This classification has been completely changed in HTML5, but the element categories above are still compatible with the new ones.

New in HTML5

The element classification used in HTML5 reflects the elements' role, rather than their appearance (inline, block).

- *Metadata* content : `<link>`, `<script>`, etc.
- *Phrasing*: ``, ``, ecc.
Similar to the "inline" class of HTML4.
- *Heading* content: `<h1>`, `<h2>`, etc.
- *Sectioning* content: `<aside>`, `<section>`, etc.
- *Flow* content: ``, `<div>`, ecc.
Similar to the homonymous class of HTML4, includes the "Phrasing" content.
- *Embedded* content: ``, `<iframe>`, `<svg>`, etc.
- *Interactive* content: `<a>`, `<button>`, `<label>`, etc.

2.4. Standard HTML attributes

- `id` : unique ID
Used to reference elements in scripts
- `class` : A list of classes
Used to assign one or more style classes to the element. The list elements are separated by spaces

- `style` : style information
Used to provide element-specific CSS style to the element
- `title` : element information
Many browsers use this attribute to generate the element tooltip
- `lang` : language code
Language codes from the I18N standard, eg. "It" or "en-us"
- `dir` : writing direction
rtl (right-to-left) or ltr (left-to-right)
- `onclick` , `ondblclick` , `onmousedown` , `onmouseup` , `onmouseover` , `onmousemove` , `onmouseout` , `onkeypress` , `onkeydown` , `onkeyup` : event handlers
Used to attach scripts to the corresponding events

New in HTML5

- `contenteditable` : marks element content as user – modifiable.
- `spellcheck` : indicates that the element content should be spell checked.
- `translate` : indicates that the element content should be translated (by automatic translators).
- `hidden` : marks the element as not relevant.
- `data-*` : user-defined extension attributes.
New attributes whose name is prefixed by "data-" can be added to any element. These attributes are used to associate data with the element, to be used by scripts in the same page.
- `role` , `aria-*` : used to give accessibility information to rich user interfaces (ARIA= Accessible Rich Internet Applications).
 - `role` indicates the element role, which may not be related to its tag name (e.g., "*checkbox*", "*grid*", "*link*", "*menu*", "*navigation*", "*form*" o "*img*": see www.w3.org/TR/wai-aria/roles).
 - `aria-*` attributes are used to support the element role with further semantic information. For example, `aria-checked` can be used on elements with "checkbox" role to indicate their state (see www.w3.org/TR/wai-aria/states_and_properties).

3. Text flow elements

3.1. Paragraphs

`<p>` : paragraph

Content: *inline*

Attributes: standard HTML

The HTML text flow is divided into paragraphs. Elements `<p>` surround each paragraph.

Each `<p>` is rendered as a separate text block, usually with a small space before and after

Empty `<p>` elements are not valid

3.2. Line breaks

`
` : line break

Content: empty

Attributes: standard HTML, `clear`

The element `
` ends the current line. The text is wrapped, without starting a new paragraph.

The `clear` attribute is used to indicate the distribution of floating objects before the line break. If present, the floats are arranged before break. *It is not allowed in HTML5.*

Browsers can add implicit `
` instead of spaces to adapt the text to the window size (word wrap). Use nonbreakable spaces () to avoid this effect.

3.3. Sections

HTML documents should be divided into logical sections.

This facilitates the rendering (especially for non-visual browsers), the search engines analysis and the conversion to other formats (e.g., PDF).

Even when using CSS to format the document, its structure should be marked by section tags, possibly modified by CSS rules, in order to be backward compatible and provide a minimal semantics to the document.

`<h1>` ... `<h6>` : section headings

Content: *inline*

Attributes: standard HTML

The elements `<hX>` create a heading of level X.

They are usually rendered in bold and with decreasing size (`<h1>` has usually double size w.r.t. the normal text)

Avoid formatting text within an `<hX>` element: instead, use styles to change the overall heading appearance.

3.4. HTML5 structural elements

HTML5 introduces several new important elements to define the document structure:

- `<main>` *encloses* the main document content.
- `<section>` *encloses* document sections. Usually contains an `<hX>` element as heading.

- `<article>` encloses an independent part of a document, such as an article in a newspaper.
- `<header>` encloses the introductory part of a section (`<section>` , `<article>` but also `<body>` or `<td>`). Often, if present, contains the section heading (`<hX>`).
- `<footer>` encloses the closing part of a section.
- `<nav>` encloses a document part used for navigational purposes (such as a menu).
- `<aside>` encloses a part of the document that is not strongly related to the rest of the document.
- `<figure>` and `<figcaption>` are used to represent a self-contained document part, such as a figure, and its caption.
- `<template>` encloses HTML fragments used as a template *by scripts* (therefore not directly shown on the page).

3.5. Containers

There are two invisible elements in HTML, which have a key role for advanced features like styles.

These two elements are `<div>` and `` . Their base semantics null: simply, `<div>` represents a block of text (but it is **not** a paragraph), while `` is a part of the text flow.

In practice, the contents of a `<div>` is preceded and followed by a carriage return, while a `` can be anywhere in the text flow.

`<div>` , `` : generic containers

Content: `` : *inline*, `<div>` : *block*

Attributes: standard HTML

These two elements define document parts to be assigned to special features, such as format, language, etc..

They are useful for the realization of complex HTML layouts, especially in conjunction with CSS.

In general, they can also be used to create ad-hoc HTML elements, with specific formatting and behavior, to complement the HTML standard elements.

The `class` and `id` attributes are important for their behavior.

4. Text formatting elements

4.1. Semantic Formatting

The *phrase elements* are used to assign a meaning (semantics) to some parts of the text.

The semantics is usually shown through a suitable rendering, but it can also be used to ease the text analysis by automated tools.

These elements have vary descriptive names: ``, ``, `<dfn>`, `<code>`, `<samp>`, `<kbd>`, `<var>`, `<cite>`, `<abbr>`, `<acronym>`

Content: *inline*

Attributes: standard HTML

The `<acronym>` element is *deprecated* in HTML5.

Base elements

- `` : Emphasis usually equivalent to italics
- `` : Strong emphasis usually equivalent to bold
- `<cite>` : External reference or citation
- `<dfn>` : Definition text not to be confused with definition lists
- `<code>` : Source code
- `<samp>` : Output example
- `<kbd>` : Keyboard-written text
- `<var>` : Variable name
- `<abbr>` : Abbreviation the *title* attribute can be used to write the complete form
- `<acronym>` : Acronym

Citations

`<blockquote>`, `<q>` : citations

Content: `<q>` : *inline*, `<blockquote>` : *block*

Attributes: standard HTML, cite

The `<blockquote>` element is used for quoting blocks of text, while `<q>` is used to include brief citations in the text flow

Both have a `cite` attribute that can be used to provide the URI of the original text

Browsers should include appropriate quotes before and after the citation

Citations inserted in a `<blockquote>` are usually indented

The use of `<blockquote>` to indent text is strongly discouraged!

Preformatted text

`<pre>` : preformatted text

Content: *inline* with some exceptions

Attributes: standard HTML

The rendering of HTML normally ignores white spaces and carriage returns in the text. The text flow follows the rules given by the corresponding elements (`<p>` , `
` ,...) and by the window size.

With the `<pre>` element the browser is required to comply with the shape given to the text in the page source: the text is rendered using a monospaced font, white space is preserved and text wrap is disabled.

Text can still contain inline HTML tags except `` , `<object>` , `<big>` , `<small>` , `<sub>` and `<sup>`

Revisions

`<ins>` , `` : inserted or deleted text

Content: *inline* or *block*

Attributes: standard HTML, cite, datetime

These elements are used to indicate text revisions.

The `cite` attribute may be used to indicate a URI where details can be found on the corresponding revision. A brief indication of the reason for the revision may also be written in the title attribute.

The `datetime` attribute can be used to write the date/time of the review

These are the only two HTML elements that can be used both inline and as blocks.

Addresses

`<address>` : Contact information

Content: *inline*

Attributes: standard HTML

This element can be used to mark the text contained in it as "contact information" for the block in which it is inserted.

Usually it is used at the body level (`<body>`) or inside a module (`<form>`).

Browsers may render the contact information in a special way, changing its position and format (e.g., always at the beginning of the block, or as a popup activated by a specific small button, etc..).

4.2. Basic Formatting

The following elements are used to apply basic formatting to characters. Their use is still tolerated, but authors are encouraged to always replace them with semantic formatting (e.g., ``) or styles (e.g., using appropriate ``)

The combination (nesting) of these elements can be exploited to achieve complex formatting (e.g., bold + italic)

The elements are: `<tt>`, `<i>`, ``, `<big>`, `<small>` **Content:** *inline*
Attributes: standard HTML

The `<big>` and `<tt>` elements are *deprecated* in HTML5.

Bold, italic,...

- `<tt>` : Monospaced text
"Teletype", usually corresponds to the courier font. Not available in HTML5.
- `<i>` : Italic
In HTML5 `<i>` elements have a new meaning and are used to represent text that has a *different quality* with respect to the surrounding one. Therefore, such text is no longer italicized by default (but CSS can be used to restore this effect).
- `` : Bold
In HTML5 `` elements are used to represent text with *higher importance* than the surrounding one.
- `<big>` : Bigger text
Not available in HTML5.
- `<small>` : Smaller text
In HTML5 `<small>` elements are used to represent side comments, such as *small prints*.

Subscript and superscript

`<sub>` , `<sup>` : superscript and subscript

Content: *inline*

Attributes: standard HTML

These elements transform the text in superscript or subscript. In other words, the font size decreases slightly and its baseline is shifted up or down with respect to the normal text

5. Lists and tables

5.1. Lists

HTML allows to define three types of lists: ordered (numbered), unordered (bulleted), and definition lists (terms or similar)

The standard rendering of lists is very simple. However, you can use style sheets to change all the rendering aspects

Lists, even of different type, can be nested

The elements used to build lists are ``, ``, ``, `<dl>`, `<dt>`, `<dd>`

Unordered (bulleted)

``, `` : unordered lists

Content: `` : one or more ``, `` : *flow*

Attributes: standard HTML

Commonly used to generate bulleted lists.

Every `` element represents a list item, and it can contain any markup, also other lists.

A list should contain at least one `` element. However, in HTML5, to avoid this common error, this constraint has been removed.

The `` element cannot be used outside lists.

Ordered (numbered)

``, `` : ordered lists

Content: `` : uno o più ``, `` : *flow*

Attributes: standard HTML

Commonly used to generate numbered lists.

Every `` element represents a list item, and it can contain any markup, also other lists. Items are progressively numbered in document order.

A list should contain at least one `` element. However, in HTML5, to avoid this common error, this constraint has been removed.

The `` element cannot be used outside lists.

Of definitions

`<dl>`, `<dt>`, `<dd>` : definition lists

Content: `<dl>` : one or more `<dt>` and `<dd>`, `<dt>` : inline, `<dd>` : *flow*

Attributes: standard HTML

Commonly used to define terms, have a great versatility

Elements `<dt>` represent the terms which are associated through a `<dd>` to a block of text that defines them

You can associate a `<dd>` to many consecutive `<dt>`

The definition `<dd>` may contain any flow element, including other lists

Usually `<dt>` are emphasized, while `<dd>` are indented

5.2. HTML5 menus

The story of an evolving element

Early HTML5 drafts described new elements like `<menu>` and `<menuitem>` that, together with attributes like *contextmenu* (on any HTML element) or *menu* (on `<button>`), allowed the to define menus (and toolbars) using a specific semantic markup.

Menu items could be created using the common `` and `<hr>` elements and with specific attributes like *type*, *label*, *icon*, *disabled*, *checked*, *radiogroup*, *default*, *command*.

However, this element seemed to be useless and a potential source of ambiguities, thus **it has been removed from the final HTML5 specification**.

Currently, even if several browsers still contain a prototypal implementation of these elements, the most correct way to define a menu structure is to use the "common" unordered lists.

5.3. Tables

HTML tables offer an extremely powerful and versatile way to organize information in rows and columns

Tables are not intended to create page layouts, but only to structure information in tabular form. Using tables to create layouts makes them not portable and is strongly discouraged.

Tables are block-type elements, so they can appear directly in the `<body>` of a document or in a `<div>` container, and should never be nested in elements such as `<p>`.

The basic element for the table definition is `<table>`

Width specifiers

The widths of the table and its columns (attribute *width*) can be specified:

- In pixels (width = "10")
- As a percentage, with respect to the space available for the table (width = "10%")

For columns only, you can specify the widths:

- Proportionally, with respect to the size required by the table (width = "10 **")
- To specify the minimum space needed for the content, use the form width = "0 ***"

If you do not specify a width:

- For a table, the width is calculated by giving the minimum width to all the columns
- For a column, the available width is proportionally distributed between all the columns without a width, always providing at least the minimum width needed by their content

Base elements

`<table>` : table definition

Content: caption (optional), sequence of col or colgroup (optional), thead (optional), tfoot (optional), tbody (implicit if not specified)

Attributes: standard HTML, cellspacing, cellpadding, width, border, rules, summary, frame

A table is defined by the elements that are nested inside the `<table>`, which must be, in this order:

- An optional caption (`<caption>`)
- An optional definition of columns/column groups
- An optional table header
- An optional table footer
- The body of the table, which contains the actual data.

The element `<caption>`, if present, has *inline* content and represents the caption of the table, which may be suitably rendered by the browser

In HTML5 the *summary* attribute is *deprecated* and the *border* attribute may only have the value "1" or be omitted.

Base attributes

`<table>` : table definition

The `cellspacing` attribute determines the space (in pixels) between cells and between the cells and the outer border of the table.

The `cellpadding` attribute determines the space (in pixels) between the border of each cell and its contents.

The `width` attribute specifies the width of the table in pixels or percentage. It is always useful to specify it, to speed up rendering (the default is "the minimum necessary").

The `border` attribute sets the thickness of the outer edge of the table. A value of zero eliminates the edge (useful to use advanced border formatting provided by CSS)

The `frame` attribute (*void, above, below, hside, vside, lhs, rhs, box, border*) determines which of the outer edges of the table will be drawn (with the thickness given by the border attribute). The default is

box.

The `rules` attribute (*none, groups, rows, cols, all*) determines which inner edges of the table (between cells) are drawn (with the thickness given by the border attribute). The default is *all*.

All these attributes are *deprecated* in HTML5 and CSS must be used instead.

Rows

`<tr>` : table rows

Content: one or more `<td>` and `<th>`

Attributes: standard HTML, `align`, `valign`

Tables are composed by a series of rows (`<tr>`), each of which contains one or more cells.

The maximum number of cells present on a single line determines the number of columns of the table. The rendering will show empty cells to the right of each line whose cells are fewer than the table columns.

The `align` attribute (*left, right, center, justify, char*) defines the horizontal alignment for all cells in the row, and `valign` (*top, bottom, middle, baseline*) defines the vertical alignment.

The *align* and *valign* attributes are *deprecated* in HTML5 and CSS must be used instead.

Cells

`<td>` , `<th>` : table cells and header cells

Content: *flow* **Attributes:** standard HTML, `align`, `valign`, `rowspan`, `colspan`, `abbr`, `axis`, `headers`, `scope`, `width`, `height`

Each row contains a set of table cells. Cells can contain arbitrary HTML content, including other tables, images, etc..

Header cells are identical to standard cells, but the browser should highlight them in some way.

Typically a cell represents the intersection of a row with a column, however, the `rowspan` and `colspan` attributes allow to specify the cell extension, respectively, in rows and columns.

Attributes `abbr` , `axis` , `headers` and `scope` are used to provide advanced data accessibility to the table.

The `width` and `height` attributes are used to provide information on cell size, but their use is **not recommended**, and should be replaced by the homonymous attributes of the `<col>` elements.

The *height*, *width*, *align*, *valign*, *abbr*, *axis* and *scope* attributes are *deprecated* in HTML5 and CSS must be used instead.

Row groups

`<thead>` , `<tbody>` , `<tfoot>` : row groups

Content: one or more `<tr>`

Attributes: standard HTML, `align`, `valign`

The rows of a table can be divided into three groups: **header** (`<thead>`), **body** (`<tbody>`) and **footer** (`<tfoot>`).

Typically, the rows in `<thead>` and `<tfoot>` are placed respectively at the beginning and at the end of the table. If the table is broken into several pages, each segment will contain the same header and footer.

If the groups are omitted, all rows are placed in an implicit `<tbody>` . Tables cannot have only a `<thead>` and/or a `<tfoot>` . If specified, these two groups should be both at the beginning of the table definition, before `<tbody>` .

In HTML5, `<tfoot>` is also allowed to appear at the end of the table definition.

Columns

`<col>` : column definition

Content: empty

Attributes: standard HTML, `align`, `valign`, `span`, `width`

Through one or more elements `<col>` **placed at the beginning of the table** (first row) it is possible to predefine the number and the characteristics of the columns that will compose the table. This facilitates the rendering of the table and decreases the code necessary to create it.

Each `<col>` represents a number of columns equal to its `span` attribute. Each column will have the size specified by `width` and the corresponding cells will have the alignment given by `align` and `valign` .

In HTML5 these elements **can only appear inside a** `<colgroup>` . Moreover, The *align*, *valign* and *width* attributes are *deprecated* in HTML5 and CSS must be used instead.

Column groups

`<colgroup>` : column group definition

Content: empty or one or more `<col>`

Attributes: standard HTML, `align`, `valign`, `span`, `width`

`<colgroup>` elements represent logical groups of columns, which can be rendered by the browser in various ways.

One or more `<colgroup>` elements can be placed at the beginning of the table (first row) as an alternative to the `<col>` elements.

An empty `<colgroup>` represents a number of columns equal to its `span` attribute. Each column will have the size specified by `width` and the corresponding cells will have the alignment given by `align` and `valign`.

To separately define the characteristics of each column in a group, it is possible to nest `<col>` elements within a `<colgroup>`. The characteristics of nested `<col>` elements (including their total number) have precedence over those specified globally by the `<colgroup>`.

6. Images and embedded objects

6.1. Images

`` : image embedding

Content: empty

Attributes: standard HTML, `src`, `alt`, `longdesc`, `width`, `height`, `ismap`, `usemap`

Inserts in the document the external image referenced by the URI in the `src` attribute

An alternative text for the image (`alt`) is an essential feature for an HTML document with high accessibility.

The `longdesc` attribute can be used to point to the URI of a document that describes in detail the image. It is *deprecated* in HTML5.

The `width` and `height` attributes should always be used to give the browser a hint about the size to be reserved for the image on the page. If these measures differ from the actual size of the image, it will be resized accordingly (and proportionally, if you specify only one attribute). In HTML5 these attributes can no longer contain percentages and cannot be used to obtain not-proportional image scaling.

6.2. Image Maps

Server-side image maps

`` : image embedding

Content: empty

Attributes: standard HTML, `src`, `alt`, `longdesc`, `width`, `height`, `ismap`, `usemap`

The boolean attribute `ismap`, if present, turns the image into a **server-side image map** when it is part of a link created by an `<a>`.

When the image is clicked, activating the link, the coordinates of the click are added to the URI specified by the `href` attribute of the tag `<a>` as a *get parameter*.

For example, if the URI is `http://test.org/test` and the coordinates `x=1, y=7`, the browser will request the resource with the URI `http://test.org/test?1,7`

Client-side image maps

`` : image embedding

Content: empty

Attributes: standard HTML, `src`, `alt`, `longdesc`, `width`, `height`, `ismap`, `usemap`

The `usemap` attribute, if present, transforms an image in a **client-side image map**.

The `usemap` attribute must contain the name of an image map defined in the same document using the element `<map>`

The image areas defined by the map will become clickable.

The client-side image maps are always preferable for accessibility reasons.

`<map>` : client side image map

Content: *block*, `<area>`

Attributes: standard HTML, `name`

The `<map>` element declares a client-side image map with the name specified by the `name` attribute.

The areas of the map can be specified by a set of elements `<area>` or `<a>`, both nested in the element `<map>`

The use of elements `<a>` is useful for creating highly accessible maps, with a rich text alternative. In this case, the tag `<a>` can be enriched with attributes such as `shape` and `coords`, which belong to the `<area>` tag. This behavior is *deprecated* in HTML5.

`<area>` : client side image map area

Content: *empty*

Attributes: standard HTML, `shape`, `coords`, `href`, `alt`

The `<area>` elements nested in a `<map>` define clickable areas of an image and their destinations

Each area has a shape determined by the attribute `shape`, i.e., *rect*, *circle* or *poly*.

The `coords` attribute contains the coordinates, separated by commas, that define the specified shape:

- For rectangles, the x and y coordinates of the top left and bottom right corner,

- For circles, the x and y coordinates of the center and the radius,
- For polygons, the x and y coordinates of all the vertices.

The link destination is specified by the `href` attribute.

A textual description of the area, specified with the `alt` attribute is mandatory for maps with high accessibility.

6.3. Objects

`<object>` : embed a generic external object

Content: *flow*, `<param>`

Attributes: standard HTML, `classid`, `codebase`, `codetype`, `data`, `type`, `standby`, `width`, `height`

The attributes `classid` and `data` may be used to indicate (in a mutually exclusive way):

- The **implementation** of the object: `classid` is a URI that points to the object to be included (e.g., applets or other small programs)
- The **data** that constitute the object: `data` is a URI pointing to the source data (e.g., images, video, audio)

To pass **parameters** to the object, you can nest `<param>` elements. The attributes `name` and `value` of these elements determine the pairs (name, value) passed to the object being initialized.

Any other content of the `<object>` element is handled by the object being loaded.

Origin and the type of an object

`<object>` : embed a generic external object

Content: *flow*, `<param>`

Attributes: standard HTML, `classid`, `codebase`, `codetype`, `data`, `type`, `standby`, `width`, `height`

The `codebase` attribute can be used to resolve relative URIs found in `classid` and `data`

`codetype` and `type` attributes specify the MIME type, respectively, of the resources pointed to by `classid` and `data`

The `standby` attribute can be used to specify a text to be displayed during the object loading.

The `width` and `height` attributes have the same purpose as in the `` element

The *classid*, *codetype*, *codebase* and *standby* attributes are *deprecated* in HTML5.

HTML5 specific tags

`<object>` tags were usually employed in HTML4 to render audio and video, or import applets.

HTML5 provides specific elements for several common uses of the `<object>` element:

- `<audio>` and `<video>` are used to import (and play) audio and/or video streams, *with a common user interface provided by the browser and a rich scripting API*.
- `<embed>` is used for plugin content.
- `<canvas>` is used to mark an area where graphics can be rendered dynamically through the element API.

7. Meta elements

7.1. Links

Relationships between documents

`<link>` : relationships between documents

Content: empty

Attributes: standard HTML, href, hreflang, type, rel, ~~rev~~, ~~charset~~, media

The element `<link>` can appear several times but only in the document `<head>`

A `<link>` does not generate a user-visible link, but declares a relation between the current document and other resources. The browser can use this information in various ways.

The `rel` (or `rev`, if the link is logically backwards) is crucial for `<link>` element, as it defines the type of relationship with the resource identified by the URI contained in the `href` attribute.

The `rev` and `charset` attributes are *deprecated* in HTML5.

`<link>` elements are used, for example, to link a document to its style sheets, to specify alternative documents in other languages, to define a logical reading sequence in a set of documents, etc..

Document relationship types

- **Start, Next, Prev:** Initial, next, previous document
Define the initial, next and previous document in the linear sequence the current document belongs to
- **Contents:** Summary
Indicates the document to be used as a summary
- **Index:** Table of Contents
Indicates the document to be used as an index for the current document
- **Glossary:** Glossary of Document
Indicates the document to use as a glossary for the current document
- **Copyright:** Copyright statement for the document

- **Chapter, Section, Subsection, Appendix:** beginning of the current chapter, section, subsection, appendix
- **Help:** Help Document
- **Bookmark:** Bookmark document
Indicates the "key" document in the collection that owns the current one, to be used to set a bookmark.
- **Alternate:** alternative document
Using the attributes *hreflang*, *media* and *type*, it is possible to define alternatives to the current document based on the language and/or reading device. A common example is the *application/rss+xml* type, which connects a web page to its *RSS feed*.
- **Stylesheet:** style sheet
Used to connect a style sheet to the document. Attributes *type* and *media* must identify respectively the MIME type of the style sheet (usually *text/css*) and the device for which it is designed. If you want to provide alternative style sheets, use the *title* attribute to give a name to the style and use the composite type "alternate stylesheet".
- **Shortcut icon :** site icon (*de facto* standard)
The image connected to a web page through this relation is used by browsers as an icon in the title bar and in the favorites list. The image type and size are subject to severe restrictions.
- **Preload:** high-priority preload of external resources
Used to preload resources, suggesting to the browser which resources on the current page are *high priority* and therefore need to be *downloaded into the local cache as soon as possible*. It is mainly used with style sheets, scripts, fonts and images. In this case, the *as* attribute is used to specify the resource type, e.g., with `as="style"` , `as="image"` , or `as="script"` . Attention: preloading does not apply the resource, but only loads it: for example, if we preload a stylesheet, we will still have to apply it by using a normal `<link>` with `rel="stylesheet"` , or if we preload a script we will then have to activate it by referencing it with a `<script>` element.
- **Prefetch:** prefetch resources useful to related pages
Prefetching is used to prefetch into the browser resources cache that are not immediately needed by the current page, but *useful for the pages linked* to it. These resources will then be downloaded, but with a *lower priority* than to the rest of the page.

Relative URI resolution

`<base>` : relative URI base

Content: empty

Attributes: standard HTML, href, target

This element, used in the document `<head>` , defines the base URI (`href` attribute) used to resolve all the relative URIs in the document

The `target` attribute may be used to define the default target in a document with frames

If `<base>` is not specified, the base URI of the current document is used to resolve all the relative URIs

7.2. Style Sheets

`<style>` : embedded style sheets

Content: text

Attributes: type, media

One or more elements `<style>` can be placed in the document `<head>`, allowing to embed one or more (fragments of) style sheets.

The `type` attribute specifies the MIME type of the style. In HTML5 this attribute can be omitted as it defaults to *text/css*.

The `media` attribute to specifies the devices for which the style has been designed.

Style sheets can also be imported from a file using the `<link>` element. In addition, you can specify a specific style for each HTML element via the *style* attribute.

The HTML attribute *class* allows to group multiple HTML elements into classes, useful to give them a uniform style.

8. Interaction elements

8.1. Links

Active links to external resources

`<a>` : active links to external resources

Content: *inline* (without nested links!)

Attributes: standard HTML, href, name, hreflang, type, rel, rev, charset, accesskey

The `<a>` element is used both to define the origin of a link or to mark link targets within a page.

- If `<a>` is a link source, the `href` attribute identifies the URI its the destination
- If `<a>` is a link target, its `name` attribute is a unique name that can be addressed by the *fragment identifier* of a URI. This allows to define links to a particular point of the target document.

The `name` attribute shares the same name space of the `id` attribute. In fact, you can use the id of any element as a *fragment identifier*. In HTML5, this is the preferred way.

The content of `<a>` may be any inline HTML, but you cannot have nested links. The default appearance of a link (underlined characters) can be varied by using style sheets.

Resource title and relationship

`<a>` : active links to other resources

Content: *inline* (without nested links!)

Attributes: standard HTML, href, name, hreflang, type, rel, ~~rev~~, ~~charset~~, accesskey

The `hreflang`, `charset` and `type` attributes can be used to give the browser information about the link destination, such as the language of the document, its character set and the type (MIME) of its content.

The `accesskey` attribute allows to specify a character that can be used as a shortcut to activate the link. It is useful for creating fast access menus and alternate link activation methods to be used when a mouse is not available.

`rel` and `rev` attributes allow to declare the relationship between the current document and the one connected by the link (see item `<link>`) considered, respectively, a forward or a backward link.

The `title` attribute can be used to better describe the link destination.

The *rev* and *charset* attributes are *deprecated* in HTML5.

8.2. Forms

Forms are (parts of) HTML documents containing, in addition to the normal markup, also special elements called **controls**, with which the user can interact.

Forms are placed in a special HTML element `<form>`. Usually, modules provide a system to send the value of their controls to the server (*submit*) for further processing.

However, there are also forms working completely on the *client side*, assisted by scripts and embedded objects.

Controls

The control elements are `<input>`, `<textarea>`, `<select>`, `<optgroup>`, `<label>`, `<fieldset>` and `<button>`.

Each control must necessarily be identified by a name, specified via the `name` attribute.

The controls may have an initial `value`, which is set when the form is created or when you *reset* it.

When the form is submitted, the server receives the pairs (name, value) of each control.

In HTML5, controls are also allowed to appear outside the `<form>` element, as long as they have a `form` attribute set to the *id* of a `<form>` placed anywhere in the document.

Base elements

`<form>` : form definition

Content: *block* (except other `<form>`), `<script>`

Attributes: standard HTML, method, action, enctype, name, accept-charset

A form requires at least specification, through the `action` attribute, of the resource URI that will process the data (e.g., a server side script)

The `method` attribute (*get* or *post*) specifies the method used to send the data to the specified resource.

If the *post* method is used, it may be necessary to specify an alternate data encoding method using the attribute `enctype` :

- The encoding *application/x-www-form-urlencoded* is the default
- The encoding *multipart/form-data* is necessary if you send files as part of the form.

The `name` attribute provides a name to the module, to be used for scripting.

The `accept-charset` attribute is often used to indicate the *encoding* of the characters sent with the form, *allowing a proper decoding on the server*.

`<input>` controls

`<input>` : form control

Content: empty

Attributes: standard HTML, type, name, value, size, maxlength, checked, disabled, readonly, src, usemap, ismap, alt

The `<input>` element is used to generate most of the form controls. The key of its versatility is the `type` attribute, which can take the following values:

- **text:** creates a line of text input
- **password:** as text, but hides the characters typed
- **checkbox:** creates a checkbox
- **radio:** creates a radio button
- **submit:** creates a button to submit the form
- **reset:** creates a button to reset the form
- **file:** create a control to upload a file
- **hidden:** creates a hidden form field
- **image:** create a control to submit the form using an image

- **button**: creates a button

HTML5 <input> controls

HTML5 introduces many other kinds of `<input>` controls, also defined by the `type` attribute:

- **tel**: a telephone number input control
- **search**: a search control
- **url**: a URL input control
- **email**: an email address input control
- **time**, **date**: specific date/time input control
- **number**: a number input control
- **range**: a range input control
- **color**: a color selection control

The *user agent* should render more appropriate form controls using this refined input type specification, to provide the user with a richer interface.

<input> controls semantics

`<input>` : form control

Content: empty

Attributes: standard HTML, type, name, value, size, maxlength, checked, disabled, readonly, src, usemap, ismap, alt

The `value` attribute provides:

- the **initialization** string for controls of type *text*, *password*, *hidden*, *file*
- The **label** for the controls of type *submit*, *reset* and *button*

The `size` attribute gives the width of the control in pixels or characters for *text* and *password* types

The `maxlength` attribute provides the maximum number of characters that can be typed in the fields of type *text* and *password*

The boolean attribute `checked` determines whether a control of type *checkbox* or *radio* is initially selected

The `src` attribute is used for *image* type controls, as well as `ismap`, `usemap` (removed from HTML5) and `alt`. The graphical buttons of type *image* send the coordinates of the click as the value (*name.x*, *name.y*) of an additional form control.

The boolean attributes `disabled` and `readonly` can be used to disable and/or make read-only the control.

HTML5 <input> controls semantics

HTML5 also introduces new `<input>` attributes:

- The `required` attribute marks the field as required. Browsers should not submit the form if these fields are not completed. This attribute can be also placed on `<select>` and `<textarea>`.
- The `min`, `max` attributes are used to define the allowed value range (for numbers, dates, etc.),
- The `autocomplete` attribute, whose value can be *on* (default) or *off*, controls the browser auto completion feature on input fields.
- The `multiple` attribute instructs the browser to allow more than one value in the field. How this behavior is rendered depends on the specific control *type*.
- The `pattern` attribute specifies a regular expression that must match the control value.
- The `step` attribute defines the allowed control values granularity.
- The `formaction`, `formmethod`, `formenctype` attributes allow to override the corresponding `<form>` attributes when the input is used as a submit button.
- The `placeholder` attribute gives an hint to be shown on the control when it is empty, and can be also placed on `<textarea>`.

<textarea> controls

`<textarea>` : form text areas

Content: text

Attributes: standard HTML, name, rows, cols, disabled, readonly

The element `<textarea>` creates a wide text input area where the user can type multiple lines of text

The visible width is determined by the attributes `rows` and `cols` (columns). The maximum number of characters can not be typed in a `<textarea>`, however, is not limited a priori.

The text nested in the element is used as its initial value.

HTML tags contained in the text are not interpreted.

<select> controls

`<select>` : multiple choice lists

Content: one or more `<option>` and `<optgroup>`

Attributes: standard HTML, name, size, multiple

The element `<select>` creates a list containing a set of options, each represented by an element `<option>`.

The `multiple` boolean attribute indicates if the user can select one or more elements of the list

The `size` attribute indicates how many options have to be displayed simultaneously in the control

The initial value and the value assigned to the control are specified by the nested `<option>` and `<optgroup>` .

Options for `<select>` controls

`<option>` , `<optgroup>` : options for `<select>` controls

Content: `<optgroup>` : one or more `<option>` , `<option>` : text

Attributes: standard HTML, label, `<option>` : selected, disabled, value

The `<option>` elements define selectable options in the `<select>` controls. The `<optgroup>` elements can be used to group together `<option>` in order to create logical structures such as menus.

The `label` attribute determines the text displayed for `<option>` and `<optgroup>` . In the case of `<option>` , you can also omit the label and specify the text to display inside the element.

The `value` attribute determines the value of the option, which will be assigned to the name of the corresponding `<select>` field during the form submission. If not specified, the contents of the option will be used as its value.

The boolean attribute `selected` determines whether the option will be initially selected.

`<button>` controls

`<button>` : form buttons

Content: *flow*, except `<a>` and all the form elements

Attributes: standard HTML, name, value, type, disabled

The `<button>` elements create buttons exactly as `<input>` elements with the corresponding `type` (which can be *submit*, *reset* or *button*)

The difference is that the content of the button is not defined by the `value` attribute, which here represents only the value given to the corresponding name when the button is pressed.

The content of the element, which can be HTML of any type and length, will be used to create the "face" of the button.

Association of text to controls

`<label>` : text associated with a control

Content: *inline*

Attributes: standard HTML, for

The `<label>` element allows to associate an *inline* text to a form control.

The associated control is identified by the value of the `for` attribute, which must correspond to the `id` (not `name` !) of one of the controls in the current form.

The browser may, for example, change the rendering of the text when the corresponding control is disabled.

You can associate multiple `<label>` to same control

Control groups

`<fieldset>` , `<legend>` : control groups

Content: `<fieldset>` : *flow*, an optional `<legend>` , `<legend>` : *inline*

Attributes: standard HTML

The `<fieldset>` elements allow to logically group parts of a form.

The `<legend>` element, if specified, provides a textual description of the `<fieldset>` .

These elements are useful to provide high accessibility to the modules and make them easier to fill.

9. References

Web Hypertext Application Technology Working Group (WHATWG)

<https://whatwg.org>

HTML 5 Specification

<http://www.w3.org/TR/html5/>

HTML 4 Specification

<http://www.w3.org/TR/html401/>

XHTML 1 Specifica

<http://www.w3.org/TR/xhtml1/>

Differences between HTML5 and HTML4

<http://www.w3.org/TR/html5-diff/>