HTML: Part II

Pau Fernández Rafa Genés Jose L. Muñoz

Universitat Politècnica de Catalunya (UPC)

Table of Contents

Global Attributes

Global attributes

All tags have a certain number of *global attributes*, common to all elements.

Global attributes:

- `class`
- `id`
- `style`
- `title`
- …

The `class` attribute

A `class` represents a set of elements in a document.

Many elements can be put in the same class.

A single element can belong to many classes.

The `class` attribute just lists all the classes of an element separated by spaces.

Setting classes

Adding one class

```
<h1 class="title">A Humble Title</h1>
```

Adding **many** classes:

```
<div class="box important left">Heads Up!</div>
<nav class="special menu left">
   <a href="#cart">Your Cart</a>
</nav>
```

(different `class` names are just separated by a space)

The `id` attribute

The `id` attribute identifies elements uniquely.

There can be no two items with the same `id`.

```
<h1 id="main-title">The Main Title</h1>
<div id="menu">
  <a href="/">Index</a>
  <a href="/help">Help</a>
</div>
```

Why do we need `class` and `id`?

To refer to elements from CSS and Javascript, we can:

- Ask for the element with a certain `id`, and apply specific styles to it, or access it programmatically.
- Ask for the *set* of elements with a certain `class`, and apply the same style to all of them, or manipulate them at once.

The **contenteditable** attribute

Setting **contenteditable** to **true**, the browser turns into an editor!

When you start editing, the browser changes the underlying DOM.

The **document.execCommand** is made available (to issue commands that manipulate the content).

Drawbacks:

- Different browsers implement edition in different ways.

https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Editable_content

Table of Contents

The DOM APIs

DOM APIs

DOM APIs allow programmers to:

- Control/access HTML elements.
- Control/access form data.
- Manipulate contents of 2D images (and `canvas` elements).
- Manage media elements (`audio` and `video`).
- Manage drag and drop events on web pages.
- Access the browser's history.

Other APIs include: Web Components, Web Storage, Web Workers, Web Sockets, Server-sent events.

The `document` object

The `document` gives access to the whole document as a Javascript object.

It belongs to the `Document` class.

Fields:

- `location`: an object which represents the current URL.
- `URL`: the current URL as a string.
- `body`: the `body` of the document (an HTMLElement).
- `head`: the `head` of the document.
- `title`: the title of the page.
- ...

Collections in the **document**

We can access many **collections** of objects within the page:

- `links`: All links (`a` elements).
- `images`: All images (`image` elements).
- `forms`: All forms (`form` elements)
- `fonts`: All fonts used by the document.
- ...

Table of Contents

Element Queries

Element queries

The `document` has methods to search elements within the page in different ways:

- `getElementsByTagName(tag)`
- `getElementById(id)`
- `getElementsByClassName(class)`
- `querySelector(selector)`
- `querySelectorAll(selector)`
- ...

getElementsByTagName

`getElementsByTagName` : all elements of a particular type of tag.

```
// Find all <h1> elements
let headers = document.getElementsByTagName('h1');
```

getElementById

`getElementById` : find the element with a certain id.

```
// Find the element with ID '#user' and hide it:
document.getElementById('user').style.display = 'none';
```

getElementsByClassName

`getElementsByClassName` : all elements with some class.

```
// Find all elements of class 'date'
let dates = document.getElementsByClassName('date');
```

querySelector

`querySelector` : find the first element matching **selector**.

```
// Get the first <span> within a <p> element
let span = document.querySelector('p span');
```

querySelectorAll

`querySelector` : find **all** elements matching **selector**.

```
// Get all <div>s with class 'comment'
let comments = document.querySelectorAll('div.comment');
```

Table of Contents

HTMLElement

The `HTMLElement` class

Every HTML element in a document has an associated Javascript object of class `HTMLElement`.

This class has fields and methods common to all HTML elements.

More specific elements inherit from this class.

Class Hierarchy

`HTMLElement` → `Element` → `Node` → `EventTarget` .

Fields of `HTMLElement`

- `innerText` : the inner text.
- `innerHTML` : the inner HTML.
- `className` : assignable list of classes (the `class` attribute).
- `title` : the `title` attribute.
- `attributes` *[read-only]*: a Map of attributes.
- `classList` *[read-only]*: list of classes.
- ...

https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement

Methods of **HTMLElement**

- `hasAttribute(attr)` : **true** if certain attribute present.
- `getAttribute(attr)` : get value of attribute.
- `setAttribute(attr)` : set value of attribute.
- `scrollIntoView(attr)` : make elem visible on the screen.
- ...

https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement

Fields of **Node**

- `firstChild` : first child.
- `lastChild` : last child.
- `childNodes` : all children.
- `parentNode` : parent of this node.
- ...

https://developer.mozilla.org/en-US/docs/Web/API/Node

Methods of **Node**

- `cloneNode([deep])` : clone the node (if **deep**, with all children).
- `appendChild(node)` : add a child to this node.
- `insertBefore(node, pos)` : insert a node before some child.
- `removeChild(node)` : remove a child from this node.
- `replaceWith(node)` : replace this node with a new one.
- All query methods of **document** are also available at every element. (They do searches in a subtree.)
- ...

https://developer.mozilla.org/en-US/docs/Web/API/Node

Table of Contents

Events

EventTarget Methods

- `addEventListener(event, handler)` : add handler for event.
- `removeEventListener(event, handler)` : remove handler for event.
- `dispatchEvent(event)` : produce a synthetic event.

https://developer.mozilla.org/en-US/docs/Web/API/EventTarget

Event list

cancel	error	show	select	scroll
mousemove	mouseenter	mouseleave	mouseup	mouseup
keyup	keydown	keypress	wheel	
cut	copy	paste	click	dblclick
blur	focus	focusin	focusout	
touchend	touchstart	touchmove	touchcancel	
submit	formdata	reset		

…

Setting a **click** event

```
<html>
  <body>
    <h1>Click me!</h1>
    <script>
      let n = 0;
      const h1 = document.querySelector('h1');
      h1.addEventListener('click', (event) => {
        n++;
        h1.innerText = `You have clicked the title ${n} times`;
      })
    </script>
  </body>
</html>
```

Event fields

Every `Element` object has fields to set event handlers.

The canonical name is **on***eventname*:

- `onmousedown`: called when a mouse button is pressed.
- `onmouseup`: called when a mouse button is released.
- `onmousemove`: called when the mouse moves over the document.
- `onwheel`: called when the mouse wheel turns.
- `onkeydown`: called when a key has been pressed.
- `onkeyup`: called when a key has been released.
- ...

Setting a **mousemove** event

```
<body>
  <p>
    Move the mouse <br>
    <span id="coords">(?, ?)</span>
  </p>
  <script>
    const coords = document.getElementById('coords');
    document.onmousemove = function (event) {
      coords.innerText = `(${event.offsetX}, ${event.offsetY})`;
    }
  </script>
</body>
```

The **Event** class

The base class for all `Event` s.

Every type of event has a specific class (`MouseEvent` , `KeyboardEvent` , `WheelEvent` , ...).

All event handlers receive an object of this type when an event occurs.

Table of Contents

Forms

The `form` tag

Forms in HTML are actually **hyperlinks** in which you can edit parameters of the URL. These parameters are specified with a `name` attribute.

To that end, a `form` element contains some `input` elements as children (the text boxes, radio buttons, etc.). These elements are often called *controls*.

Attributes:

- `action` The URL to submit to.
- `method` the HTTP method to use (”get” or ”post”)

The `form` submission

When you press the ”submit” button, the browser loads the URL passing in the data in the form controls

```
<!-- page is "http://task-manager.com" -->
<form action="/addtask" method="get">
  <p><label>Task <input type="text" name="task"></label></p>
  <p><label>Minutes <input type="number" name="min"></label></p>
  <input type="submit" value="Add Task">
</form>
```

If the task is ”Kill Bill” with 90 minutes, the loaded URL will be

```
https://task-manager.com/addtask?task=Kill+Bill&min=90
```

The `event.preventDefault` method

Inside an event handler, you can prevent the default behavior by calling `preventDefault()` on the event:

```
const form = document.forms[0];
form.addEventListener('submit', (event) => {
  event.preventDefault(); // <-- Do not submit the form
  console.log("The form was stopped it");
})
```

Form structure

Each part of a form is considered a paragraph.

```
<form method="get" action="https://google.com">
  <p>...</p>
  <p>...</p>
</form>
```

The `input` tag

The `input` tag represents a large family of form controls: text boxes, buttons, checkboxes, radio buttons, etc.

General attributes:

- `type` : The type of control.
- `name` : Name of the element in the form submission.
- `value` : Initial value of the control.

Text boxes

With `type` to "`text`" or "`search`", the `input` is a text box.

Attributes:

- `required` : The value is required for the submission.
- `minlength` : Minimum length *(enforced by **`required`**)*
- `maxlength` : Maximum length *(You can't write more)*
- `size` : Size of the control *(in characters)*
- `placeholder` : Hint text *(as an example for the user)*

```
<input type="text" minlength="4" maxlength="25"
       size="15" required>
```

The `label` tag

The `label` tag marks a caption on a user interface.

The `for` attribute links a `label` with the corresponding control in the form.

```
<label for="name">Enter your name:</label>
```

Associating a **label** with an **input** (1)

Mode 1: Making the `input` a child of the `label`

```
<form action="...">
  <label>
    Username
    <input type="text" name="user" required>
  </label>
  ...
</form>
```

Associating a **label** with an **input** (2)

Mode 2: When the two elements are separated, you can explicitly mention the `id` of the `input` in the `for` attribute in the `label`

```
<form action="...">
  <label for="username">Username</label>
  ...
  <input type="text" id="username" name="user" required>
</form>
```

Why associate a `label` with an `input`?

If you click the `label`, the `input` will be focused.

For accessibility: screen readers will read the `label` before reading the `input`.

Lesson: giving more semantic information allows the browser to do its job!

Types of text boxes

Different `type`s of text boxes:

- `tel` : editing a telephone *(free-form really)*
- `password` : editing a password *(characters hidden)*
- `url` : editing a URL *(URL checked on submit)*
- `email` : editing a URL *(URL checked on submit)*
- `number` : editing a number *(only digits, shows arrows)*

Table of Contents

Form Controls

Buttons

An `input` tag of `type`

- `submit` : Submits the form *(default text: "Submit")*
- `reset` : Resets the form *(default text: "Reset")*
- `button` : Does nothing *(unless we respond to events)*

Attributes:

- `value` : Text inside the button.

```
<form>
  <input type="submit">
  <input type="reset" value="Clear">
  <input type="button" value="I do nothing">
</form>
```

The **button** tag

A `button` tag shows a button and:

- Behaves like a submit `input` within a form.
- Behaves like a normal button outside a form (does nothing).

```
<button>This is a normal button</button>
<form action="/submit" method="get">
  <button>This is a submit button</button>
</form>
```

Sliders

An `input` with `type=range` shows a slider.

Attributes:

- `min` : Minimim value *(default = 0)*
- `max` : Maximum value *(default = 100)*
- `step` : Quantization *(the value is a multiple of the **`step`**)*

```
<form>
  <input type="range" id="volume" name="volume"
         min="0" max="11">
  <label for="volume">Volume</label>
</form>
```

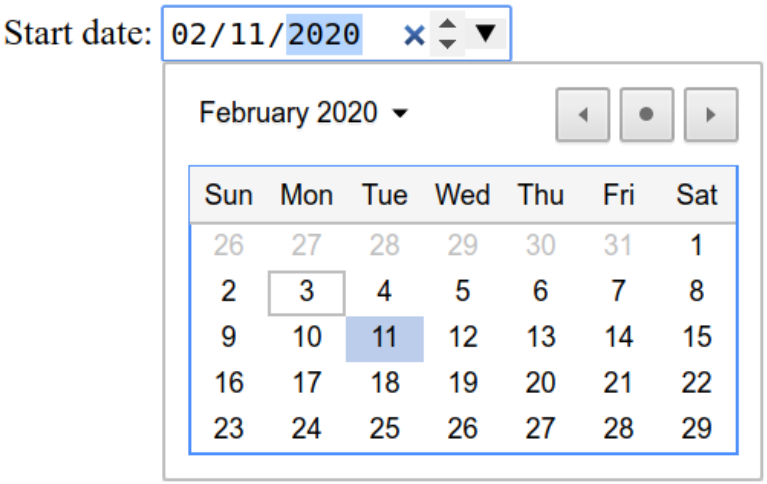
Color Pickers

An `input` with `type=color` will show a color picker as a button.


```
<input type="color" value="2020-02-11">
```

Date and time pickers

An `input` with `type=date` edits a date (with arrows and calendar). Attributes: `min`, `max`.




```
<input type="date" value="2020-02-11"
       min="2020-01-02" max="2020-12-31">
```

One radio button

An `input` tag with `type=radio` shows only a small radio button.

A `label` marks its associated text.

Attributes:

- `name` editing property
- `value` resulting value if selected

```
<label>
  <input type="radio" name="color" value="red">
  Red
</label>
```

Radio button group

To make radio button groups, use the same `name` in all of them.

```
<label>
  <input type="radio" name="color" value="red">
  Red
</label>
<label>
  <input type="radio" name="color" value="blue">
  Blue
</label>
<label>
  <input type="radio" name="color" value="yellow">
  Yellow
</label>
```

Checkboxes

An `input` tag with `type=checkbox` shows only a small checkbox.

A `label` marks its associated text.

Attributes:

- `name` editing property
- `checked` initial value of the checkbox (`true` or `false`).

```
<label>
  <input type="radio" name="married" checked="on"> Bachelor
</label>
```

The **select** tag

The `select` tag shows a dropdown list with several option. The `name` attribute indicates the `form` property.

The `option` tag marks all possible options.

```
<select name="color">
  <option value="red">Red</option>
  <option value="yellow">Yellow</option>
  <option value="blue">Blue</option>
  <option value="green">Green</option>
</select>
```

The default selection

To show a message asking to make a selection it is typical to use an `option` with an empty value.

```
<select name="weapon" required>
  <option value="">Select a weapon</option>
  <option value="gun">Gun</option>
  <option value="bow">Bow</option>
  <option value="stone">Stone</option>
  <option value="poisin">Poison</option>
</select>
```

*Adding the **required** attribute ensures a value is selected.*

Omitting the **option** close tag

Within a `select` you can omit `option` closing tags.

```
<select name="weapon" required>
  <option value="">Select a weapon
  <option value="gun">Gun
  <option value="bow">Bow
  <option value="stone">Stone
  <option value="poisin">Poison
</select>
```

Text Area

The `textarea` tag shows a multiline plain text edit control.

The contents of the element are the control's default value.

Attributes:

- `readonly` : the content can't be edited.
- `rows` : number of rows to show.
- `cols` : maximum number of characters per line.

```
<textarea name="config">
DB_FILE=/Users/pauek/db/test.db
DB_USER=pauek
DB_PASSWORD=123456
</textarea>
```

Links

HTML Elements Reference
https://developer.mozilla.org/en-US/docs/Web/HTML/Element

HTML Specification for Developers
https://html.spec.whatwg.org/dev/