HTML5 Form Notes – Part A

How to structure an HTML form

While building an HTML form, one of the most important steps is to structure it correctly. It is important for mainly two reasons: firstly, to guarantee that your form will be usable and secondly, to make it accessible (that is, usable by differently-abled people). Accessibility of HTML forms is essential, and we will see how to make a form accessible. It's not difficult, but there are a few tricks you need to know.

The flexibility of HTML forms makes them one of the most complex structures in HTML . By mixing dedicated form elements and attributes you can build any kind of basic form. It's true that there are some richer form technologies such as XForms (now obsolete), but those kinds of forms are, unfortunately, not widely implemented by all browsers. Because of this, most of the time we have to rely on JavaScript to enhance HTML forms. In this article, we will cover in details, how to use all the HTML form elements. If you want to know more about building custom form widgets, you can read the article How to build custom form widgets.

Global structure

The <form> element

The `<form>` element is the element that formally defines a form, and its attributes, that determine the way this form will behave. Each time you want to create an HTML form, you must start it by using this element. Many assistive technologies or browser plugins can discover `<form>` elements and can implement special hooks to make them easier to use.

Note: It's strictly forbidden to nest a form inside another form. Doing so can behave in an unpredictable way that will depend on which browser the user is using.

The `<form>` element support the following attributes, all of which are optional:

Attributes for the `<form>` element

Attribute name	Default value	Description
`accept-charset`	`UNKNOWN`	A space-delimited list of character encodings that the server accepts. The default value is the special string `UNKNOWN`, and in that case the encoding corresponds to the encoding of the document containing the form element.
`action`		The URI of a web page that processes the information submitted via the form.

Attributes for the `<form>` element

Attribute name	Default value	Description
`autocomplete`	`on`	Indicates whether widgets in this form can have their values automatically completed by the browser by default. This attribute can take two values: `on` or `off`.
`enctype`	`application/x-www-form-urlencoded`	When the value of the `method` attribute is `post`, this attribute is the MIME type of content that is used to submit the form to the server. Possible values are: `application/x-www-form-urlencoded` `multipart/form-data`: Use this value if you are using an `<input>` element with the `type` attribute set to *file*. `text/plain`
`method`	`get`	The HTTP method that the browser uses to submit the form. This attribute can take two values: `get` or `post`.
`name`		The name of the form. It must be unique among the forms in a document and may not be an empty string. You should usually use the `id` attribute instead.
`novalidate`	(*false*)	This Boolean attribute indicates that the form is not to be validated after submission.
`target`	`_self`	A name or keyword indicating where to display the response that has been received after submitting the form; this can be an `<iframe>`, tab, or window, for example. The following keywords are available as possible values for this attribute: `_self`: Load the response into the same browsing context (`<iframe>`, tab, window, etc.) as the current one. `_blank`: Load the response into a new browsing context. `_parent`: Load the response into the parent browsing context of the current one. If there is no parent, this option behaves the same way as `_self`. `_top`: Load the response into the top-level browsing context (that is, the browsing context that is an ancestor of the current one, and has no parent). If there is no parent, this option behaves the same way as `_self`.

Note that it's always possible to use a form widget outside of a `<form>` element but if you do so, that form widget has nothing to do with any form. Such widgets can be used outside a form, but then you should have a special plan for such widgets, since they will do nothing on their own. You will have to customize its behavior with JavaScript.

Technically speaking, HTML5 introduces the `form` attribute on HTML form elements. It should let you explicitly bind an element with a form even if it is not enclosed within a `<form>`. Unfortunately, for the time being, the implementation of this feature across browsers is not yet good enough to rely on it.

The <fieldset> and <legend> elements

The `<fieldset>` element is a convenient way to create groups of widgets that share the same purpose. A `<fieldset>` element can be labeled with a `<legend>`element. The `<legend>` element formally describes the purpose of the `<fieldset>` element. Many assistive technologies will use the `<legend>` element as if it is a part of the label of each widget inside the corresponding `<fieldset>` element. For example, some screen readers such as Jaws or NVDA will pronounce the legend's content before pronouncing the label of each widget.
Here is a little example:

```
<form>
  <fieldset>
    <legend>Fruit juice size</legend>
    <p>
      <input type="radio" name="size" id="size_1" value="small" />
      <label for="size_1">Small</label>
    </p>
    <p>
      <input type="radio" name="size" id="size_2" value="medium" />
      <label for="size_2">Medium</label>
    </p>
    <p>
      <input type="radio" name="size" id="size_3" value="large" />
      <label for="size_3">Large</label>
    </p>
  </fieldset>
</form>
```

With this example, a screen reader will pronounce "Fruit juice size small" for the first widget, "Fruit juice size medium" for the second, and "Fruit juice size large" for the third.

The use case in this example is one of the most important. Each time you have a set of radio buttons, you need to make sure that they are nested inside a`<fieldset>` element. There are other use cases, and in general the `<fieldset>` element can also be used to section a form. Because of its influence over assistive technology, the `<fieldset>` element is one of the key elements for building accessible forms; however it is your responsibility not to abuse it. If possible, each time you build a form, try to listen to how a screen reader interprets it. If it sounds odd, try to improve the form structure.

The `<fieldset>` element supports the following specific attributes:

Attributes for the `<fieldset>` element

Attribute name	Default value	Description
`disabled`	(*false*)	If this Boolean attribute is set, then the descendent form controls (other than descendants of its first optional `<legend>`element) are disabled and not editable. They won't receive any browsing events, like mouse clicks or focus-related ones. Often browsers display such controls as gray.

The <label> element

The `<label>` element is the formal way to define a label for an HTML form widget. This is the most important element if you want to build accessible forms.
The `<label>` element supports the following attributes:

Attributes for the `<label>` element

Attribute name	Default value	Description
`for`		The ID of a label able widget in the same document as the `<label>` element. The first such element in the document with an ID matching the value of the `for` attribute is the labeled widget for this label element.

A `<label>` element is bound to its widget with the `for` attribute. The `for` attribute references the `id` attribute of the corresponding widget. A widget can be nested inside its `<label>` element but even in that case, it is considered best practice to set the `for` attribute because some assistive technologies do not understand implicit relationships between labels and widgets.
Note that even without considering assistive technologies, having a formal label set for a given widget lets users click the label to activate the corresponding widget in all browsers. This is especially useful for radio buttons and checkboxes.

```
<form>
  <p>
    <input type="checkbox" id="taste_1" name="taste_cherry" value="1">
    <label for="taste_1">I like cherry</label>
  </p>
  <p>
    <label for="taste_2">
      <input type="checkbox" id="taste_2" name="taste_banana" value="1">
      I like banana
    </label>
  </p>
</form>
```

Some assistive technologies can have trouble handling multiple labels for a single widget. Because of this, you should nest a widget inside its corresponding element to build an accessible form.

Let's consider this example:

```
<form>
  <p>Required fields are followed by <strong><abbr
title="required">*</abbr></strong>.</p>

  <!-- when the thing you are labeling is a descendant of the label, it is not necessary
to use the 'for' attribute on the label. -->
  <!-- So this: -->
  <label>
    <span>Name: </span>
    <input type="text" name="username" required />
    <strong><abbr title="required">*</abbr></strong>
  </label>

  <!-- is the same as this: -->
  <div>
    <label for="username">Name: </label>
    <input id="username" type="text" name="username" required />
    <strong><abbr title="required">*</abbr></strong>
  </div>

  <p>
    <label for="birth"> <!-- so here, the 'for' attribute is redundant. -->
      <span>Date of birth: </span>
      <input type="text" id="birth" name="userbirth" maxlength="10" /> <em>formated as
mm/dd/yyyy</em>
    </label>
  </p>
</form>
```

In this example, the first paragraph defines the rule for required elements. It must be at the beginning to be sure that assistive technologies such as screen readers will display or vocalize it to the user before he finds a required element. That way, he always knows what he has to do.

The first field is required, so its label element indicates both its name and the fact that it's a required field. That way, a screen reader will vocalize the label as "*Name star*" or "*Name required*" (this depends on the screen reader's settings, but it's always consistent with what was vocalized in the first paragraph). If you used two labels, there would be no guarantee that the user would be informed that this element was required.
The second form element works similarly. By using this technique, you can be sure that the user is told how to format the date when entering it.

The <output> element

This element is used to store the output of a calculation. It formally defines a relationship between the fields used to get the data required to perform the calculation and an element to be used to display the results. It is also understood as a live region by some assistive technologies (which means that when the content of the `<output>` element changes, the assistive technology is aware of that change and can react to it).
The `<output>` element supports the following attributes:

Attributes for the `<output>` element

Attribute name	Default value	Description
for		A space-delimited list of IDs of other elements, indicating that those elements contribute input values to (or otherwise affect) the calculation.

Common HTML structures used with forms

Beyond the structures specific to HTML forms, it's good to remember that forms are just HTML. This means that you can use all the power of HTML to structure an HTML form.

As you can see in the examples, It's common practice to wrap a label and its widget with a `<p>` or a `<div>` element.
In addition to the `<fieldset>` element, it's also common practice to use HTML titles and sectioning to structure a complex form.
HTML lists are also often used when using checkboxes and radio buttons.

Let's see an example of a simple payment form:

```
<form>
  <h1>Payment form</h1>
  <p>Required fields are followed by <strong><abbr
title="required">*</abbr></strong>.</p>

  <section>
    <h2>Contact information</h2>

    <fieldset>
      <legend>Title</legend>
      <ul>
        <li>
          <label for="title_1">
            <input type="radio" id="title_1" name="title" value="M." />
            Mister
          </label>
        </li>
        <li>
```

```
          <label for="title_2">
            <input type="radio" id="title_2" name="title" value="Ms." />
            Miss
          </label>
        </li>
      </ul>
    </fieldset>

    <p>
      <label for="name">
        <span>Name: </span>
        <input type="text" id="name" name="username" required />
        <strong><abbr title="required">*</abbr></strong>
      </label>
    </p>

     <p>
      <label for="mail">
        <span>E-mail: </span>
        <input type="email" id="mail" name="usermail" required />
        <strong><abbr title="required">*</abbr></strong>
      </label>
    </p>
  </section>

  <section>
    <h2>Payment information</h2>

    <p>
      <label for="card">
        <span>Card type:</span>
        <select id="card" name="usercard">
          <option value="visa">Visa</option>
          <option value="mc">Mastercard</option>
          <option value="amex">American Express</option>
        </select>
      </label>
    </p>
    <p>
      <label for="number">
        <span>Card number:</span>
        <input type="text" id="number" name="cardnumber" required />
        <strong><abbr title="required">*</abbr></strong>
      </label>
    </p>
    <p>
      <label for="date">
        <span>Expiration date:</span>
        <input type="text" id="date" name="expiration" required />
        <strong><abbr title="required">*</abbr></strong>
        <em>formated as mm/yy</em>
      </label>
    </p>
  </section>

  <section>
    <p>
```

```
      <button>Validate the payment</button>
    </p>
  </section>
</form>
```

References: "How to Structure an HTML Form." How to Structure an HTML Form. Mozilla Developer Network, 03 Mar. 2016. Web. 15 Mar. 2016. Contributors List Bellow…

Contributors: rolfedh, PushpitaPikuDey, marcos-abreu, Qombella, susanBuck, Sheppy, rylan, andrew-luhring, LayZeeDK, xfq, Jeremie, Serenity, kscarfone, FredB, notabene