mdn web docs

How to structure a web form

With the basics out of the way, we'll now look in more detail at the elements used to provide structure and meaning to the different parts of a form.

Prerequisites:	Basic computer literacy, and a basic understanding of HTML.
Objective:	To understand how to structure HTML forms and give them semantics so they are usable and accessible.

The flexibility of forms makes them one of the most complex structures in HTML; you can build any kind of basic form using dedicated form elements and attributes. Using the correct structure when building an HTML form will help ensure that the form is both usable and accessible.

The <form> element

The `<form>` element formally defines a form and attributes that determine the form's behavior. Each time you want to create an HTML form, you must start it by using this element, nesting all the contents inside. Many assistive technologies and browser plugins can discover `<form>` elements and implement special hooks to make them easier to use.

We already met this in the previous article.

> Warning: It's strictly forbidden to nest a form inside another form. Nesting can cause forms to behave unpredictably, so it is a bad idea.

It's always possible to use a form control outside of a `<form>` element. If you do so, by

default that control has nothing to do with any form unless you associate it with a form using its `form` attribute. This was introduced to let you explicitly bind a control with a form even if it is not nested inside it.

Let's move forward and cover the structural elements you'll find nested in a form.

The <fieldset> and <legend> elements

The `<fieldset>` element is a convenient way to create groups of widgets that share the same purpose, for styling and semantic purposes. You can label a `<fieldset>` by including a `<legend>` element just below the opening `<fieldset>` tag. The text content of the `<legend>` formally describes the purpose of the `<fieldset>` it is included inside.

Many assistive technologies will use the `<legend>` element as if it is a part of the label of each control inside the corresponding `<fieldset>` element. For example, some screen readers such as Jaws and NVDA will speak the legend's content before speaking the label of each control.

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>
```

> Note: You can find this example in fieldset-legend.html (see it live also).

When reading the above form, a screen reader will speak "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 should nest them inside a `<fieldset>` element. There are other use cases, and in general the `<fieldset>` element can also be used to section a form. Ideally, long forms should be spread across multiple pages, but if a form is getting long and must be on a single page, putting the different related sections inside different fieldsets improves usability.

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 <label> element

As we saw in the previous article, 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 — when implemented properly, screen readers will speak a form element's label along with any related instructions, as well as it being useful for sighted users. Take this example, which we saw in the previous article:

```
<label for="name">Name:</label> <input type="text" id="name" name="user_name" />
```

With the `<label>` associated correctly with the `<input>` via its `for` attribute (which contains the `<input>` element's `id` attribute), a screen reader will read out something like "Name, edit text".

There is another way to associate a form control with a label — nest the form control within the `<label>`, implicitly associating it.

```
<label for="name">
  Name: <input type="text" id="name" name="user_name" />
</label>
```

Even in such cases however, it is considered best practice to set the `for` attribute to ensure all assistive technologies understand the relationship between label and widget.

If there is no label, or if the form control is neither implicitly nor explicitly associated with a label, a screen reader will read out something like "Edit text blank", which isn't very helpful at all.

Labels are clickable, too!

Another advantage of properly set up labels is that you can click or tap the label to activate the corresponding widget. This is useful for controls like text inputs, where you can click the label as well as the input to focus it, but it is especially useful for radio buttons and checkboxes — the hit area of such a control can be very small, so it is useful to make it as easy to activate as possible.

For example, clicking on the "I like cherry" label text in the example below will toggle the selected state of the taste_cherry checkbox:

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

> Note: You can find this example in checkbox-label.html (see it live also).

Multiple labels

Strictly speaking, you can put multiple labels on a single widget, but this is not a good idea as some assistive technologies can have trouble handling them. In the case of multiple labels, you should nest a widget and its labels inside a single `<label>` element.

Let's consider this example:

```
<p>Required fields are followed by <span aria-label="required">*</span>.</p>

<!-- So this: -->
<!--div>
  <label for="username">Name:</label>
  <input id="username" type="text" name="username">
  <label for="username"><span aria-label="required">*</label>
</div-->

<!-- would be better done like this: -->
<!--div>
  <label for="username">
    <span>Name:</span>
    <input id="username" type="text" name="username">
    <span aria-label="required">*</span>
  </label>
</div-->

<!-- But this is probably best: -->
<div>
  <label for="username">Name: <span aria-label="required">*</span></label>
  <input id="username" type="text" name="username" />
</div>
```

The paragraph at the top states a rule for required elements. The rule must be included before it is used so that sighted users and users of assistive technologies such as screen readers can learn what it means before they encounter a required element. While this helps inform users what an asterisk means, it can not be relied upon. A screen reader will speak an asterisk as "star" when encountered. When hovered by a sighted mouse user, "required" should appear, which is achieved by use of the `title` attribute. Titles being read aloud depends on the screen reader's settings, so it is more reliable to also include the `aria-label` attribute, which is always read by screen readers.

The above variants increase in effectiveness as you go through them:

- In the first example, the label is not read out at all with the input — you just get "edit text blank", plus the actual labels are read out separately. The multiple `<label>` elements confuse the screen reader.
- In the second example, things are a bit clearer — the label read out along with the input is "name star name edit text required", and the labels are still read out separately. Things are still a bit confusing, but it's a bit better this time because the `<input>` has a label associated with it.
- The third example is best — the actual label is read out all together, and the label read out with the input is "name required edit text".

> Note: You might get slightly different results, depending on your screen reader. This was tested in VoiceOver (and NVDA behaves similarly). We'd love to hear about your experiences too.

> Note: You can find this example on GitHub as required-labels.html (see it live also). Don't test the example with 2 or 3 of the versions uncommented — screen readers will definitely get confused if you have multiple labels AND multiple inputs with the same ID!

Common HTML structures used with forms

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

As you can see in the examples, it's common practice to wrap a label and its widget with a `<li>` element within a `<ul>` or `<ol>` list. `<p>` and `<div>` elements are also commonly used. Lists are recommended for structuring multiple checkboxes or radio buttons.

In addition to the `<fieldset>` element, it's also common practice to use HTML titles (e.g. h1, h2) and sectioning (e.g. `<section>`) to structure complex forms.

Above all, it is up to you to find a comfortable coding style that results in accessible, usable forms. Each separate section of functionality should be contained in a separate `<section>` element, with `<fieldset>` elements to contain radio buttons.

Active learning: building a form structure

Let's put these ideas into practice and build a slightly more involved form — a payment form. This form will contain a number of control types that you may not yet understand. Don't worry about this for now; you'll find out how they work in the next article (Basic native form controls). For now, read the descriptions carefully as you follow the below instructions, and start to form an appreciation of which wrapper elements we are using to structure the form, and why.

1. To start with, make a local copy of our blank template file and the CSS for our payment form in a new directory on your computer.

2. Apply the CSS to the HTML by adding the following line inside the HTML `<head>`:

```
<link href="payment-form.css" rel="stylesheet" />
```

3. Next, create your form by adding a `<form>` element:

```
<form>
...
</form>
```

4. Inside the `<form>` element, add a heading and paragraph to inform users how required fields are marked:

```
<h1>Payment form</h1>
<p>
  Required fields are followed by
  <strong><span aria-label="required">*</span></strong>.
</p>
```

5. Next, we'll add a larger section of code into the form, below our previous entry. Here you'll see that we are wrapping the contact information fields inside a distinct `<section>` element. Moreover, we have a set of three radio buttons, each of which we are putting inside its own list (`<li>`) element. We also have two standard text `<input>` s and their associated `<label>` elements, each contained inside a `<p>` , and a password input for entering a password. Add this code to your form:

```
<section>
  <h2>Contact information</h2>
  <fieldset>
    <legend>Title</legend>
    <ul>
      <li>
        <label for="title_1">
          <input type="radio" id="title_1" name="title" value="K" />
          King
        </label>
      </li>
      <li>
```

```
        <label for="title_2">
          <input type="radio" id="title_2" name="title" value="Q" />
          Queen
        </label>
      </li>
      <li>
        <label for="title_3">
          <input type="radio" id="title_3" name="title" value="J" />
          Joker
        </label>
      </li>
    </ul>
  </fieldset>
  <p>
    <label for="name">
      <span>Name: </span>
      <strong><span aria-label="required">*</span></strong>
    </label>
    <input type="text" id="name" name="username" />
  </p>
  <p>
    <label for="mail">
      <span>Email: </span>
      <strong><span aria-label="required">*</span></strong>
    </label>
    <input type="email" id="mail" name="usermail" />
  </p>
  <p>
    <label for="pwd">
      <span>Password: </span>
      <strong><span aria-label="required">*</span></strong>
    </label>
    <input type="password" id="pwd" name="password" />
  </p>
</section>
```

6. The second `<section>` of our form is the payment information. We have three distinct controls along with their labels, each contained inside a `<p>`. The first is a drop-down menu (`<select>`) for selecting credit card type. The second is an `<input>` element of

type `tel`, for entering a credit card number; while we could have used the `number` type, we don't want the number's spinner UI. The last one is an `<input>` element of type `text`, for entering the expiration date of the card; this includes a placeholder attribute indicating the correct format, and a pattern that tests that the entered date has the correct format. These newer input types are reintroduced in The HTML5 input types. Enter the following below the previous section:

```
<section>
  <h2>Payment information</h2>
  <p>
    <label for="card">
      <span>Card type:</span>
    </label>
    <select id="card" name="usercard">
      <option value="visa">Visa</option>
      <option value="mc">Mastercard</option>
      <option value="amex">American Express</option>
    </select>
  </p>
  <p>
    <label for="number">
      <span>Card number:</span>
      <strong><span aria-label="required">*</span></strong>
    </label>
    <input type="tel" id="number" name="cardnumber" />
  </p>
  <p>
    <label for="expiration">
      <span>Expiration date:</span>
      <strong><span aria-label="required">*</span></strong>
    </label>
    <input
      type="text"
      id="expiration"
      required="true"
      placeholder="MM/YY"
      pattern="^(0[1-9]|1[0-2])\/([0-9]{2})$" />
  </p>
</section>
```

7. The last section we'll add is a lot simpler, containing only a `<button>` of type `submit`, for submitting the form data. Add this to the bottom of your form now:

```
<section>
  <p>
    <button type="submit">Validate the payment</button>
  </p>
</section>
```

8. Finally, complete your form by adding the outer `<form>` closing tag:

```
</form>
```

You can see the finished form in action below (also find it on GitHub — see our payment-form.html source and running live):

Test your skills!

You've reached the end of this article, but can you remember the most important information? You can find a further test to verify that you've retained this information before you move on — see Test your skills: Form structure.

Summary

You now have all the knowledge you'll need to properly structure your web forms. We will cover many of the features introduced here in the next few articles, with the next article

looking in more detail at using all the different types of form widgets you'll want to use to collect information from your users.

See also

- A List Apart: Sensible Forms: A Form Usability Checklist

Advanced Topics

- How to build custom form controls
- Sending forms through JavaScript
- Property compatibility table for form widgets

This page was last modified on Feb 22, 2023 by MDN contributors.