



Your first HTML form

English ▼

Overview: Forms

Next

The first article in our series provides your very first experience of creating an HTML form, including designing a simple form, implementing it using the right HTML form controls and other HTML elements, adding some very simple styling via CSS, and how data is sent to a server.

Prerequisites: Basic computer literacy, and a basic understanding of [HTML](#).

Objective: To gain familiarity with what HTML forms are, what they are used for, how to think about designing them, and the basic HTML elements you'll need for simple cases.

What are HTML forms?

HTML Forms are one of the main points of interaction between a user and a web site or application. Forms allow users to enter data, generally sending that data to the web server, but a web page can also use form data client side.

An HTML Form is made of one or more widgets. Those widgets can be single or multi-line text fields, select boxes, buttons, checkboxes, or radio buttons. The single line text field widgets can even require data entered to be of a specific format or value. Form widgets should be paired with a properly implemented label that describes their purpose — labels help instruct both sighted and blind users on what to enter into a form input.

The main difference between a HTML form and a regular HTML document is that most of the time, the data collected by the form is sent to a web server. In that case, you need to set up a web server to receive and process the data. How to set up such a server is beyond the scope of this article, but if you want to know more, see [Sending form data](#) later in the module.

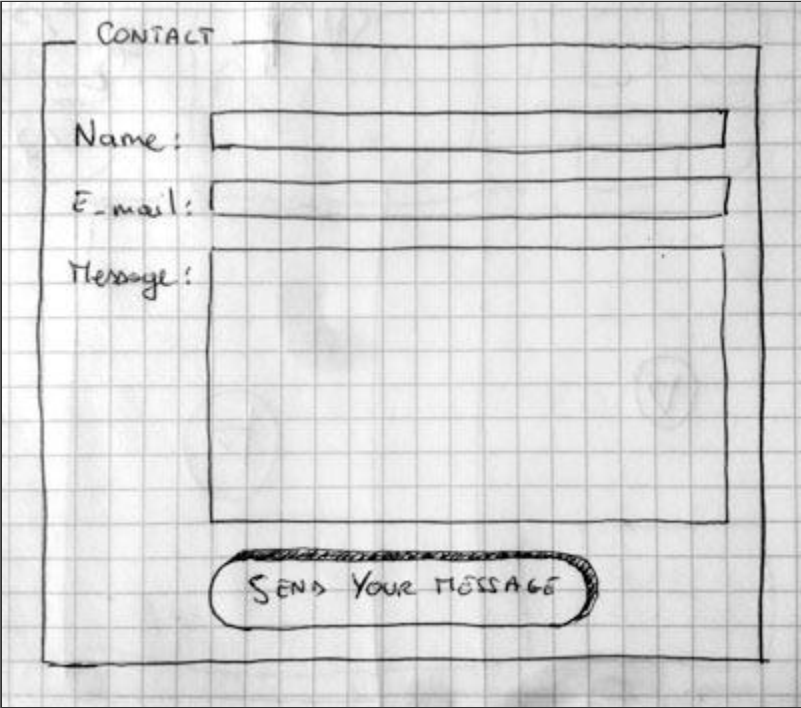
Designing your form

Before starting to code, it's always better to step back and take the time to think about your form. Designing a quick mockup will help you to define the right set of data you want to ask your user. From a user experience (UX) point of view, it's important to remember that the bigger your form, the more you risk losing users. Keep it simple and stay focused: ask only for that data you absolutely need.

Designing forms is an important step when you are building a site or application. It's beyond the scope of this article to cover the user experience of forms, but if you want to dig into that topic you should read the following articles:

- [Smashing Magazine](#) has some good articles about forms UX, including an older but still relevant [Extensive Guide To Web Form Usability](#) article.
- [UXMatters](#) is also a very thoughtful resource with good advice from basic best practices to complex concerns such as multi-page forms.

In this article, we'll build a simple contact form. Let's make a rough sketch.



A hand-drawn sketch of a contact form on graph paper. The form is titled "CONTACT" at the top. It contains three input fields: "Name:", "E-mail:", and "Message:". The "Message:" field is a larger text area. At the bottom of the form is a button labeled "SEND YOUR MESSAGE".

Our form will contain three text fields and one button. We are asking the user for their name, their e-mail and the message they want to send. Hitting the button will send their data to a web server.

Active learning: Implementing our form HTML

Ok, now we're ready to go to HTML and code our form. To build our contact form, we will use the following HTML elements: `<form>`, `<label>`, `<input>`, `<textarea>`, and `<button>`.

Before you go any further, make a local copy of our simple HTML template — you'll enter your form HTML into here.

The `<form>` element

All HTML forms start with a `<form>` element like this:

```
1 | <form action="/my-handling-form-page" method="post">
2 |
3 | </form>
```

This element formally defines a form. It's a container element like a `<div>` or `<p>` element, but it also supports some specific attributes to configure the way the form behaves. All of its attributes are optional but it's considered best practice to always set at least the `action` attribute and the `method` attribute.

- The `action` attribute defines the location (URL) where the form's collected data should be sent when it is submitted.
- The `method` attribute defines which HTTP method to send the data with (it can be "get" or "post").

Note: If you want to dig into how those attributes work, it is detailed in the [Sending form data](#) article.

For now, add the above `<form>` element into your HTML body.

The `<label>`, `<input>`, and `<textarea>` elements

Our contact form is really simple: the data entry portion contains three text fields, each with a label. The input field for the name is a basic single-line text field, the input field for the e-mail is a single-line text field that accepts only e-mail addresses, and the input field for the message is a multiline text field.

In terms of HTML code we need something like the following to implement these form widgets:

```
1  <form action="/my-handling-form-page" method="post">
2    <div>
3      <label for="name">Name:</label>
4      <input type="text" id="name" name="user_name">
5    </div>
6    <div>
7      <label for="mail">E-mail:</label>
8      <input type="email" id="mail" name="user_mail">
9    </div>
10   <div>
11     <label for="msg">Message:</label>
12     <textarea id="msg" name="user_message"></textarea>
13   </div>
14 </form>
```

Update your form code to look like the above.

The `<div>` elements are there to conveniently structure our code and make styling easier (see later in the article). Note the use of the *for* attribute on all `<label>` elements; it's a formal way to link a label to a form widget. This attribute references the `id` of the corresponding widget. There is great benefit to doing this: it associates to label with the form control enabling mouse, trackpad and touch device users to click on the label to activate the corresponding widget, and provides an accessible name for screen readers. If you want a better understanding of the other benefits of this attribute, you can find the details in [How to structure an HTML form](#).

On the `<input>` element, the most important attribute is the `type` attribute. This attribute is extremely important because it defines the way the `<input>` element appears and behaves. You'll find more about this in the [native form widgets](#) article later on.

- In our simple example, we use the value `text` for the first input — the default value for this attribute. It represents a basic single-line text field that accepts any kind of text input.

- For the second input, we use the value `email` that defines a single-line text field that only accepts a well-formed e-mail address. This turns a basic text field into a kind of "intelligent" field that will perform some checks on the data typed by the user. It also provides for the inclusion of the `@` symbol on the default keyboard on devices with dynamic keyboards. You'll find out more about form validation in the [Form data validation](#) article later on.

Last but not least, note the syntax of `<input>` vs. `<textarea></textarea>`. This is one of the oddities of HTML. The `<input>` tag is an empty element, meaning that it doesn't need a closing tag. `<textarea>` is not an empty element, meaning it should be closed with the proper ending tag. This has an impact on a specific feature of HTML forms: the way you define the default value. To define the default value of an `<input>` element you have to use the `value` attribute like this:

```
1 | <input type="text" value="by default this element is filled with this te
```

If you want to define a default value for a `<textarea>`, you just have to put that default value between the opening and closing tag of the `<textarea>` element, like this:

```
1 | <textarea>by default this element is filled with this text</textarea>
```

The `<button>` element

The markup for our form is almost complete; we still have to add a button to allow the user to send, or "submit", their data once they have filled out the form. This is done by using the `<button>` element; add the following just above the closing `</form>` tag:

```
1 | <div class="button">
2 |   <button type="submit">Send your message</button>
3 | </div>
```

You'll see that the `<button>` element also accepts a `type` attribute — this accepts one of three values: `submit`, `reset`, or `button`.

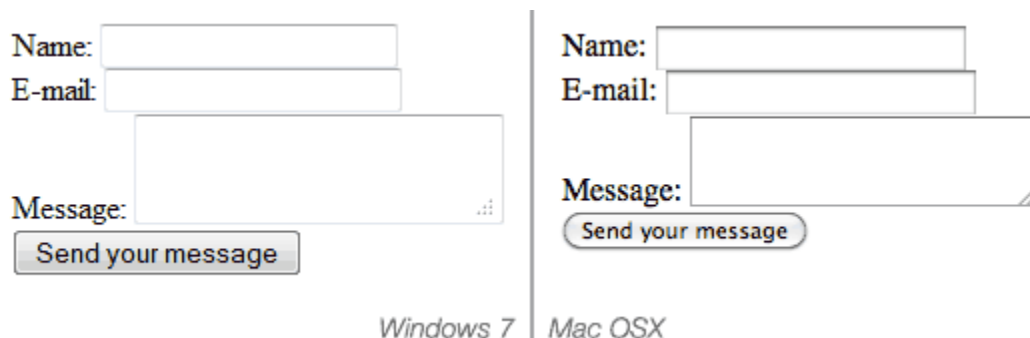
- A click on a `submit` button (the default value) sends the form's data to the web page defined by the `action` attribute of the `<form>` element.

- A click on a `reset` button resets all the form widgets to their default value immediately. From a UX point of view, this is considered bad practice.
- A click on a `button` button does... nothing! That sounds silly, but it's amazingly useful for building custom buttons with JavaScript.

Note: You can also use the `<input>` element with the corresponding `type` to produce a button, for example `<input type="submit">`. The main advantage of the `<button>` element is that the `<input>` element only allows plain text as its label whereas the `<button>` element allows full HTML content, allowing more complex, creative button text.

Basic form styling

Now that you have finished writing your form's HTML code, try saving it and looking at it in a browser. At the moment, you'll see that it looks rather ugly.



Note: If you don't think you've got the HTML code right, try comparing it with our finished example — see `first-form.html` (also see it live).

Forms are notoriously tricky to style nicely. It is beyond the scope of this article to teach you form styling, so for the moment, we will just get you to add some CSS to make it look OK.

First of all, add a `<style>` element to your page, inside your HTML head. It should look like so:

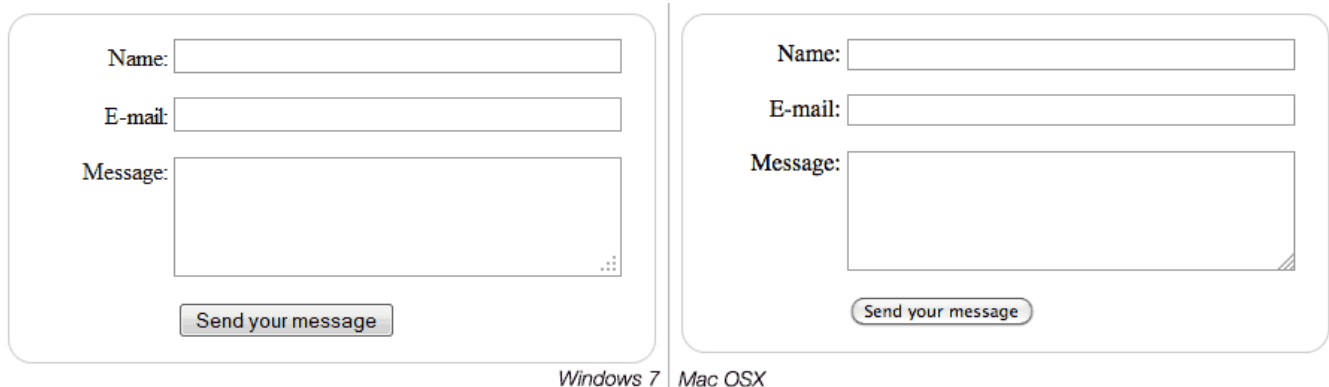
```
1 <style>
2
3 </style>
```

Inside the style tags, add the following CSS, just as shown:

```
1  form {
2    /* Center the form on the page */
3    margin: 0 auto;
4    width: 400px;
5    /* Form outline */
6    padding: 1em;
7    border: 1px solid #CCC;
8    border-radius: 1em;
9  }
10
11 form div + div {
12   margin-top: 1em;
13 }
14
15 label {
16   /* Uniform size & alignment */
17   display: inline-block;
18   width: 90px;
19   text-align: right;
20 }
21
22 input,
23 textarea {
24   /* To make sure that all text fields have the same font settings
25      By default, textareas have a monospace font */
26   font: 1em sans-serif;
27
28   /* Uniform text field size */
29   width: 300px;
30   box-sizing: border-box;
31
32   /* Match form field borders */
33   border: 1px solid #999;
34 }
35
36 input:focus,
37 textarea:focus {
38   /* Additional highlight for focused elements */
39   border-color: #000;
40 }
41
42 textarea {
```

```
43  /* Align multiline text fields with their labels */
44  vertical-align: top;
45
46  /* Provide space to type some text */
47  height: 5em;
48  }
49
50  .button {
51    /* Align buttons with the text fields */
52    padding-left: 90px; /* same size as the label elements */
53  }
54
55  button {
56    /* This extra margin represent roughly the same space as the space
57       between the labels and their text fields */
58    margin-left: .5em;
59  }
```

Now our form looks much less ugly.



Note: You can find it on GitHub at [first-form-styled.html](#) (also see it live).

Sending form data to your web server

The last part, and perhaps the trickiest, is to handle form data on the server side. The `<form>` element defines where and how to send the data thanks to the `action` and `method` attributes.

We provide a name to each form control. The names are important both browser and server sides; they tell the browser which name to give each piece of data and, on the server side, they

let the server handle each piece of data by name. The form data is sent to the server as name/value pairs.

To name the data in a form you need to use the `name` attribute on each form widget that will collect a specific piece of data. Let's look at some of our form code again:

```
1  <form action="/my-handling-form-page" method="post">
2    <div>
3      <label for="name">Name:</label>
4      <input type="text" id="name" name="user_name" />
5    </div>
6    <div>
7      <label for="mail">E-mail:</label>
8      <input type="email" id="mail" name="user_email" />
9    </div>
10   <div>
11     <label for="msg">Message:</label>
12     <textarea id="msg" name="user_message"></textarea>
13   </div>
14
15   ...
```

In our example, the form will send 3 pieces of data named "user_name", "user_email", and "user_message". That data will be sent to the URL "/my-handling-form-page" using the HTTP POST method.

On the server side, the script at the URL "/my-handling-form-page" will receive the data as a list of 3 key/value items embodied in the HTTP request. The way this script will handle that data is up to you. Each server-side language (PHP, Python, Ruby, Java, C#, etc.) has its own mechanism. It's beyond the scope of this guide to go deeply into that subject, but if you want to know more, we have provided some examples in the [Sending form data](#) article.

Summary

Congratulations, you've built your first HTML form. It looks like this live:



A screenshot of a web form. It is enclosed in a light gray rounded rectangle. Inside, there are three labels with corresponding input fields: 'Name:' followed by a single-line text input, 'E-mail:' followed by a single-line text input, and 'Message:' followed by a multi-line text area. At the bottom of the form is a button labeled 'Send your message'.

That's only the beginning, however — now it's time to take a deeper look. HTML forms are way more powerful than what we saw here and the other articles of this guide will help you to master the rest.

[Overview: Forms](#)[Next](#)

In this module

- [Your first HTML form](#)
- [How to structure an HTML form](#)
- [The native form widgets](#)
- [Sending form data](#)
- [Form data validation](#)
- [How to build custom form widgets](#)
- [Sending forms through JavaScript](#)
- [HTML forms in legacy browsers](#)
- [Styling HTML forms](#)
- [Advanced styling for HTML forms](#)
- [Property compatibility table for form widgets](#)

Last modified: Oct 5, 2019, by MDN contributors

Related Topics

Complete beginners start here!

- ▶ [Getting started with the Web](#)

HTML — Structuring the Web

- ▶ [Introduction to HTML](#)

- ▶ [Multimedia and embedding](#)

- ▶ [HTML tables](#)

- ▼ [HTML forms](#)

[HTML forms overview](#)

[Your first HTML form](#)

[How to structure an HTML form](#)

[The native form widgets](#)

[Sending form data](#)

[Form validation](#)

[How to build custom form widgets](#)

[Sending forms through JavaScript](#)

[HTML forms in legacy browsers](#)

[Styling HTML forms](#)

[Advanced styling for HTML forms](#)

[Property compatibility table for form widgets](#)

CSS — Styling the Web

- ▶ [CSS first steps](#)

- ▶ [CSS building blocks](#)

- ▶ [Styling text](#)

- ▶ [CSS layout](#)

JavaScript — Dynamic client-side scripting

- ▶ JavaScript first steps
- ▶ JavaScript building blocks
- ▶ Introducing JavaScript objects
- ▶ Asynchronous JavaScript
- ▶ Client-side web APIs

Accessibility — Make the web usable by everyone

- ▶ Accessibility guides
- ▶ Accessibility assessment

Tools and testing

- ▶ Cross browser testing

Server-side website programming

- ▶ First steps
- ▶ Django web framework (Python)
- ▶ Express Web Framework (node.js/JavaScript)

Further resources

- ▶ Common questions

How to contribute



Learn the best of web development

Get the latest and greatest from MDN delivered straight to your inbox.

Sign up now