

2.8: Forms

Forms

Forms drive the internet. They are perhaps the most critical element in creating an interactive experience for your end users, and allow you to take in input. Forms define places on a page where the user's interaction can add, change, interact with, or remove the data in your system. The actions and fields you allow in your form determine what the user is allowed to do, and what information he is allowed to see.

Form elements range from username and password style boxes to large text fields, drop down lists, checkboxes and more. All of the elements within a form block are sent from that page to the destination attribute of the form declaration, called an action. When the user hits send this information is then made available in one of several ways to the receiving page or script.

To create a form section, we provide the form with a name, id, action, and method. An example with blank attributes looks like this:

```
1. <form name="" id="" action="" method=""></form>
```

Our form's name and id are how we will refer to it in our code when interacting with it using CSS, JavaScript, or other languages. The action is where the page should send the information (and where the browser will go when we hit send). Our method is how we will send our information, using either GET or POST.

Get

Sending the data using the get method places all of the form fields by name and value (called a key and value pair) into the address bar, making our URI longer by appending each item to the receiving page's address. An easy way to remember this is that the user "gets" to see the information that was sent, as it will appear in the address bar at the top of the browser. The benefit of using the get method is that the destination can be bookmarked with the data that was sent. So, if your form is used to search a library and filter results, you could save the result as a bookmark, and return to the page in the future, seeing the same results without filling out the form again.

While beneficial, there are two instances in which we DO NOT want to use get: either we do not want the user (or anyone) to see what was sent, such as passwords or confidential information, or we want to send a lot of information. There is a practical limit to how much data can reliably and safely be passed using get, although no formal ceiling. The practical limits are those created by the browser or server's ability to store the information being sent. When developing large get requests, determine which browsers you want to support, and how old, and figure out which of the oldest has the lowest maximum threshold.

Post

Posted data is sent from the browser to the server in the background, as the client and server first begin to talk. The data is sent in the headers (see) of the communication, and are not visible to the end user. Pages bookmarked with the post method will not have access to the information later on, and that information is lost if the user leaves the page.

How the data is used and values or new content returned bring us to scripting languages. Skip to the server-side language section of this text to learn about that process.

Form Fields

When a webpage with a form is rendered, we can identify a specific field for the user to start with. You may have experienced this in action when you load a website and find the cursor already in a textbox. This is autofocus. To include this function, simply add the attribute autofocus to the field the user will want or need first. We can also apply placeholders to our text fields that tell the user what we want them to enter with the placeholder attribute. To begin, we will add a text input inside our form tags for a name field:

```
1. <form name="" id="" action="" method="">
2. <input type="text" placeholder="Your First Name" autofocus name="name" />
3. </form>
```

Many of the new elements of HTML5 we look at will also assist us with our validation tasks as users fill out forms. These inputs will attempt to validate and/or limit user entry to only valid data. By doing this immediately, we create a better experience for both the user and the programmer. Traditionally, validation had to be done when data was sent to the server, resulting in the page

reloading if there were errors. The other popular approach was to perform validation using JavaScript on the client-side (avoiding the reload), but validation would still have to be repeated on the server in the event the end user had JavaScript disabled. Some of the more useful input types are the following:

`<input type="url">` Will attempt to format the user's text into a proper link, or display an error. `<input type="email">` Will make sure an email entered is in proper format, or display an error.

We can also create an input that limits values to a fixed range and increment limitations, which we used to have to display to the user on the page, and then validate after entry:

1. `<input type="range" min="10" max="50" step="5" value="30">`

These limits on a range (shown as a slider) also are valid on a number field as well (shown as arrows):

1. `<input type="number" min="10" max="50" step="5" value="30">`

HTML5 also introduces a wealth of calendar and time controls. We can specify a date, week, or month as well as a time, day and time, and local day and time. Each of these fields will limit the user's entry to valid fields for that type.

Calendar options:

1. `<input type="date" name="date"/>`
2. `<input type="week" name="week"/>`
3. `<input type="month" name="month"/>`

Time Options:

1. `<input type="time" name="time"/>`
2. `<input type="datetime" name="dateTime"/>`
3. `<input type="datetime-local" name="localDateTime"/>`

Learn more

Keywords, search terms: Tables, forms

Do not Use Tables For Layout: <http://webdesign.about.com/od/layout/a/aa111102a.htm>

Mozilla HTML Forms Guide <https://developer.mozilla.org/en-US/...ide/HTML/Forms>

2.8: Forms is shared under a [not declared](#) license and was authored, remixed, and/or curated by LibreTexts.

- 2.8: Forms by [Michael Mendez](#) has no license indicated.