**Joel Ndiba Mwaura**

**Applied Technology BYU-Idaho**

**WDD330 Web Frontend ||**

**May 24, 2023**

**Week-06 Notes**

**Form properties and methods**

Forms and control elements, such as <input> have a lot of special properties and events.

Working with forms will be much more convenient when we learn them.

[**Navigation: form and elements**](https://javascript.info/form-elements#navigation-form-and-elements)

Document forms are members of the special collection document.forms.

That’s a so-called *“named collection”*: it’s both named and ordered. We can use both the name or the number in the document to get the form.

Form navigation:

**document.forms**

A form is available as document.forms[name/index].

**form.elements**

Form elements are available as form.elements[name/index], or can use just form[name/index]. The elements property also works for <fieldset>.

**element.form**

Elements reference their form in the form property.

Value is available as input.value, textarea.value, select.value, etc. (For checkboxes and radio buttons, use input.checked to determine whether a value is selected.)

For <select>, one can also get the value by the index select.selectedIndex or through the options collection select.options.

These are the basics to start working with forms. We’ll meet many examples further in the tutorial.

In the next chapter we’ll cover focus and blur events that may occur on any element, but are mostly handled on forms.

**Forms: event and method submit**

The submit event triggers when the form is submitted, it is usually used to validate the form before sending it to the server or to abort the submission and process it in JavaScript.

The method form.submit() allows to initiate form sending from JavaScript. We can use it to dynamically create and send our own forms to server.

Let’s see more details of them.

[**Event: submit**](https://javascript.info/forms-submit#event-submit)

There are two main ways to submit a form:

1. The first – to click <input type="submit"> or <input type="image">.
2. The second – press Enter on an input field.

Both actions lead to submit event on the form. The handler can check the data, and if there are errors, show them and call event.preventDefault(), then the form won’t be sent to the server.

In the form below:

1. Go into the text field and press Enter.
2. Click <input type="submit">.

[**Method: submit**](https://javascript.info/forms-submit#method-submit)

To submit a form to the server manually, we can call form.submit().

Then the submit event is not generated. It is assumed that if the programmer calls form.submit(), then the script already did all related processing.

Sometimes that’s used to manually create and send a form, like this:

let form = document.createElement('form');

form.action = '<https://google.com/search>';

form.method = 'GET';

form.innerHTML = '<input name="q" value="test">';

// the form must be in the document to submit it document.body.append(form);

form.submit();

*From <*[*https://javascript.info/forms-submit*](https://javascript.info/forms-submit)*>*

[*Using FormData Objects Effectively*](https://www.youtube.com/watch?v=GWJhE7Licjs)



**FormData**

This chapter is about sending HTML forms: with or without files, with additional fields and so on.

[FormData](https://xhr.spec.whatwg.org/#interface-formdata) objects can help with that. As you might have guessed, it’s the object to represent HTML form data.

The constructor is:

letformData =newFormData([form]);

If HTML form element is provided, it automatically captures its fields.

The special thing about FormData is that network methods, such as fetch, can accept a FormData object as a body. It’s encoded and sent out with Content-Type: multipart/form-data.

From the server point of view, that looks like a usual form submission.

[**Sending a simple form**](https://javascript.info/formdata#sending-a-simple-form)

Let’s send a simple form first.

As you can see, that’s almost one-liner:

<formid="formElem"><inputtype="text"name="name"value="John"><inputtype="text"name="surname"value="Smith"><inputtype="submit"></form><script>  
 formElem.onsubmit=async(e)=>{e.preventDefault();letresponse =awaitfetch('/article/formdata/post/user',{method:'POST', body:newFormData(formElem)});letresult =awaitresponse.json();alert(result.message);};</script>

*From <*[*https://javascript.info/formdata*](https://javascript.info/formdata)*>*

[FormData](https://xhr.spec.whatwg.org/#interface-formdata) objects are used to capture HTML form and submit it using fetch or another network method.

We can either create new FormData(form) from an HTML form, or create an object without a form at all, and then append fields with methods:

* formData.append(name, value)
* formData.append(name, blob, fileName)
* formData.set(name, value)
* formData.set(name, blob, fileName)

Let’s note two peculiarities here:

1. The set method removes fields with the same name, append doesn’t. That’s the only difference between them.
2. To send a file, 3-argument syntax is needed, the last argument is a file name, that normally is taken from user filesystem for <input type="file">.

Other methods are:

* formData.delete(name)
* formData.get(name)
* formData.has(name)

*From <*[*https://javascript.info/formdata*](https://javascript.info/formdata)*>*

**Client-side form validation**

* [Previous](https://developer.mozilla.org/en-US/docs/Learn/Forms/UI_pseudo-classes)
* [Overview: Forms](https://developer.mozilla.org/en-US/docs/Learn/Forms)
* [Next](https://developer.mozilla.org/en-US/docs/Learn/Forms/Sending_and_retrieving_form_data)

Before submitting data to the server, it is important to ensure all required form controls are filled out, in the correct format. This is called **client-side form validation**, and helps ensure data submitted matches the requirements set forth in the various form controls. This article leads you through basic concepts and examples of client-side form validation.

[**What is form validation?**](https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation#what_is_form_validation)

Go to any popular site with a registration form, and you will notice that they provide feedback when you don't enter your data in the format they are expecting. You'll get messages such as:

* "This field is required" (You can't leave this field blank).
* "Please enter your phone number in the format xxx-xxxx" (A specific data format is required for it to be considered valid).
* "Please enter a valid email address" (the data you entered is not in the right format).
* "Your password needs to be between 8 and 30 characters long and contain one uppercase letter, one symbol, and a number." (A very specific data format is required for your data).

This is called **form validation**. When you enter data, the browser and/or the web server will check to see that the data is in the correct format and within the constraints set by the application. Validation done in the browser is called **client-side** validation, while validation done on the server is called **server-side** validation. In this chapter we are focusing on client-side validation.

If the information is correctly formatted, the application allows the data to be submitted to the server and (usually) saved in a database; if the information isn't correctly formatted, it gives the user an error message explaining what needs to be corrected, and lets them try again.

We want to make filling out web forms as easy as possible. So why do we insist on validating our forms? There are three main reasons:

* **We want to get the right data, in the right format.** Our applications won't work properly if our users' data is stored in the wrong format, is incorrect, or is omitted altogether.
* **We want to protect our users' data**. Forcing our users to enter secure passwords makes it easier to protect their account information.
* **We want to protect ourselves**. There are many ways that malicious users can misuse unprotected forms to damage the application. See [Website security](https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Website_security).  
  **Warning:** Never trust data passed to your server from the client. Even if your form is validating correctly and preventing malformed input on the client-side, a malicious user can still alter the network request.

**WEEK 06 QUATIONS**

My VC code is not working with Git? What should I do?

*From <*[*https://developer.mozilla.org/en-US/docs/Learn/Forms/Form\_validation*](https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation)*>*

**Form Code pen link**

<https://codepen.io/mwajoendi/pen/zYmyKvx>