**CSS3 Pseudo-Classes and HTML5 Forms**

Contrary to what [HTML5 Please](http://html5please.com/) and the W3C would have you believe, CSS3 is not part of HTML5. “But this is HTML5 Doctor,” I imagine you saying. “Why are you talking about CSS3 here?” Well, I want to talk about a very specific part of CSS3, one that works in perfect tandem with HTML5, specifically with the new form functions that are available.

One of the killer features that HTML5 introduces is client-side form validation without using JavaScript. Now, I know the other doctors haven’t discussed this yet, and as I’m just a locum, you might feel as if I’m putting the cart before the horse (#mixedmetaphors), but I’m hoping to explain just enough to let you know how it works, and then your regular doctors can provide the details at a later date.

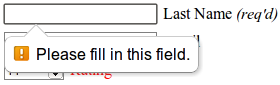
## A Recap of Pseudo-Classes [#](http://html5doctor.com/css3-pseudo-classes-and-html5-forms/#pseudo-classes)

A **pseudo-class** is information about an element that’s in the document tree but not available through any specified attributes. For example, the :first-child pseudo-class applies to elements which are the first child of their parents. You can get that information from the DOM tree, but there’s no first-child attribute.

In CSS2.1, there were a handful of pseudo-classes available, notably the link states (:link, :visited) and those of user actions (:active, :hover). In CSS3, that handful becomes a basketful, with pseudo-classes of structure (:nth-child, :nth-of-type), UI element (:enabled, :checked), and the ones we’ll look at now, of form validation.

## Form Validation without JavaScript [#](http://html5doctor.com/css3-pseudo-classes-and-html5-forms/#form-validation)

So as I mentioned at the beginning, HTML5 introduces client-side form validation without the need for JavaScript. When you attempt to submit a form, your browser will validate all of the fields and return an error if any of the fields is incorrect. This will happen mostly automagically — provided your browser has the capability — and will look something like what’s shown in Figure 1:

Figure 1: A form validation error message

Those error messages are browser and OS specific, and hard to modify ([which I documented on my blog](http://www.broken-links.com/2011/06/16/styling-html5-form-validation-errors/)), but you can change the way the errors appear on the elements themselves with the new validation pseudo-classes, which are part of the [CSS3 Basic UI module](http://www.w3.org/TR/css3-ui/).

I’ll cover current browser support as we go through the article, but in a nutshell:

* Firefox and IE10 support some of it.
* Opera and Chrome support all of it.
* Safari supports all of it too, but form validation is disabled by default.

That being the case, I’d recommend using Chrome, Opera, or Safari to view the examples I’ll be showing you.

If you want to keep up to date with browser support for form validation, I suggest keeping an eye on [When can I use](http://caniuse.com/#feat=form-validation).

So with all of the boring technical guff out of the way, let’s get on with the interesting technical guff and take a look at some examples of form validation.

### Required and Optional Elements [#](http://html5doctor.com/css3-pseudo-classes-and-html5-forms/#required-and-optional-elements)

One of the most common patterns of validation is that of mandatory (that is, required) values — the fields that the user must complete in order to progress. To mark a form element as mandatory, you need only add the required attribute to it:

<input type="text" required>

If you want to apply styles to this in order to show that requirement, you can use the new :required pseudo-class, which applies rules to any matched element which has the required attribute. For example, you may want to put an asterisk after the text of the label:

input:required + label::after { content: "\*"; }

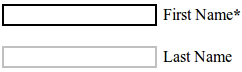
The exact rules you use depends on how you’ve marked up your document. The rule in this example is based on the markup having this structure:

<input type="text" required id="foo">  
<label for="foo">Foo</label>

The opposite of required is, of course, optional. And wouldn’t you know it, we have a pseudo-class for that too, logically named :optional. It affects any form element that doesn’t have a required attribute. For example, you may want to make those have a lighter border:

input:optional { border-color: silver; }

You can see this in action in Figure 2, or [take a look at a live example](http://html5doctor.com/demos/pseudo-classes/example1.html).

Figure 2: Input fields styled with :required (top) and :optional (bottom)

### Valid and Invalid Elements [#](http://html5doctor.com/css3-pseudo-classes-and-html5-forms/#valid-and-invalid-elements)

There are other types of validation besides simply “required” or “optional”. You can use pattern matching, such as email address validation:

<input type="email">

If the user enters anything into the field, it has to match the pattern of an email address, or else it will be marked as invalid. (Of course, if it’s not required, then it can be left blank.)

You can style valid or invalid form elements using pseudo-classes called — wait for it — :valid and :invalid. Maybe you want add a symbol depending on the validation status:

input:invalid + label::after { content: ' ⨉'; }  
input:valid + label::after { content: ' ✓'; }

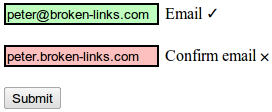
Note that these will take effect as soon as the page loads, so you’ll probably want to use JavaScript to apply them only after submission, perhaps by adding a class to the form with JavaScript:

document.forms[].addEventListener('submit', function(e) {  
  e.currentTarget.classList.add('submitted');  
  // Your fallback form validation function  
});

The exact script you use will vary, but however you do it, you can use that class to style the form elements, like so:

.submitted input:invalid + label::after { content: ' ⨉'; }

Figure 3 shows the output of this, and you can also [see the live example](http://html5doctor.com/demos/pseudo-classes/example2.html).

Figure 3: Showing the use of the :valid (top) and :invalid (bottom) pseudo-classes

Two extra things to note about this example: first, in order to use my own JavaScript, I’ve prevented the form from automatically validating by using the novalidate attribute on the submit button; and second, Firefox puts a coloured glow around elements depending on their validation state, which I’ve over-ruled by using their proprietary :-moz-ui-invalid selector. View the source of the demo to see how these are used.

### Number Ranges [#](http://html5doctor.com/css3-pseudo-classes-and-html5-forms/#number-ranges)

Some of the new input types, such as number, allow a range of values using the min and max attributes, like so:

<input type="number" max="10" min="1">

Although form controls will usually prevent the user from entering a value outside of this range, there may be occasions (such as when JavaScript is used to provide values) where the value provided to the input does exceed the range:

<input type="number" max="10" min="1" value="11">

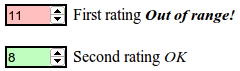
When this happens, you can use the :out-of-range pseudo-class for your styling:

input[type='number']:out-of-range { border-color: red; }

And, as you’d expect, there’s a counterpart known by the name of :in-range:

input[type='number']:in-range { border-color: green; }

In Figure 4 you can see these two pseudo-classes at work, and once more there’s also [a live example](http://html5doctor.com/demos/pseudo-classes/example3.html) to view.

Figure 4: The :out-of-range (top) and :in-range (bottom) pseudo-classes applied to a range input

### Reading and Writing

You may have a form field that’s pre-filled with a value that you don’t want the user to be able to edit, such as terms and conditions in a textarea. If that’s the case, you can add the readonly attribute:

<textarea readonly>Lorem ipsum</textarea>

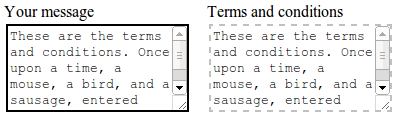
You can style elements which have the readonly attribute applied, using…can you guess? You are correct, imagined reader: the :read-only pseudo-class. You could combine it with the user-select CSS property to prevent the user from being able to select the text (in the example, I’ve also changed the border style to make it more obvious):

textarea:read-only { user-select: none; }

And if you should need it, there is of course a :read-write pseudo-class for elements which don’t have the attribute applied:

textarea:read-write { user-select: text; }

You can see the results in Figure 5, and as before there’s [a live example](http://html5doctor.com/demos/pseudo-classes/example4.html) too.

Figure 5: The textarea on the left is styled using :read-write, and the one on the right, with :read-only

Firefox and IE10 have actually implemented the readonly attribute, but don’t yet have support for these pseudo-classes.

## Browser Support [#](http://html5doctor.com/css3-pseudo-classes-and-html5-forms/#browser-support)

This table lists support for HTML5 Form Validation in browsers at the time of writing:

| Browser/Attribute | Chrome | Firefox | IE | Opera | Safari |
| --- | --- | --- | --- | --- | --- |
| :optional / :required | 10.0 | 4.0 | 10.0 | 10.0 | 5.0 |
| :invalid / :valid | 10.0 | 4.0 | n/a | 10.0 | 5.0 |
| :in-range / :out-of-range | 10.0 | n/a | n/a | 10.0 | 5.0 |
| :read-only / :read-write | 10.0 | n/a | n/a | 10.0 | 5.0 |
| *Browser support for HTML5 Form Validation* | | | | | |

## That’s All from Me [#](http://html5doctor.com/css3-pseudo-classes-and-html5-forms/#goodbye)

I hope I’ve given you enough to whet your appetite for finding out more about HTML5 Form Validation. There’s plenty more to it, including a full JavaScript API (you can [read about this in the HTML5 spec](http://dev.w3.org/html5/spec-author-view/constraints.html#the-constraint-validation-api), but ).

Thanks for your time, and thanks to the Doctors for this opportunity.