mdn web docs

Other form controls

We now look at the functionality of non- `<input>` form elements in detail, from other control types such as drop-down lists and multi-line text fields, to other useful form features such as the `<output>` element (which we saw in action in the previous article), and progress bars.

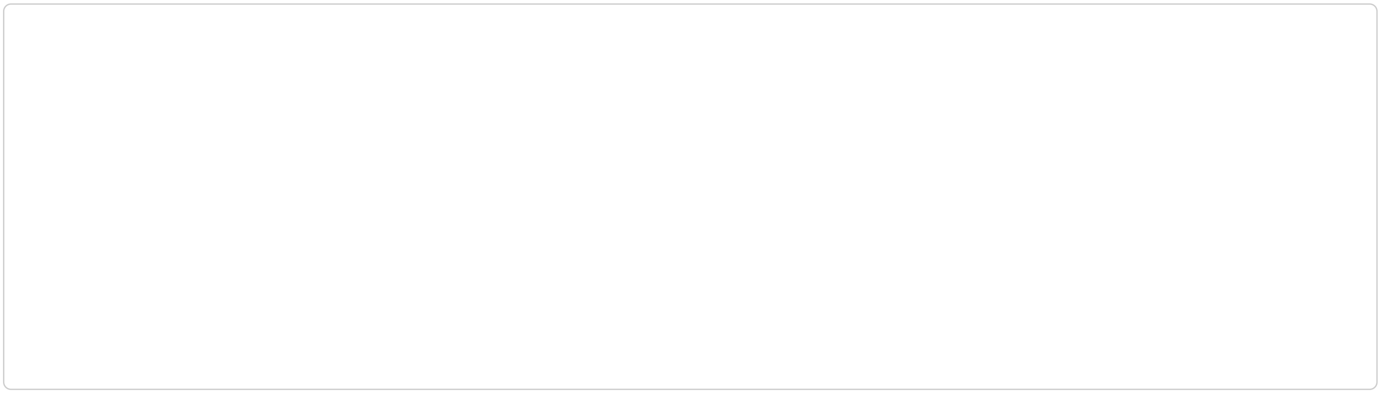
Prerequisites:	Basic computer literacy, and a basic understanding of HTML.
Objective:	To understand the non- `<input>` form features, and how to implement them using HTML.

Multi-line text fields

A multi-line text field is specified using a `<textarea>` element, rather than using the `<input>` element.

```
<textarea cols="30" rows="8"></textarea>
```

This renders like so:



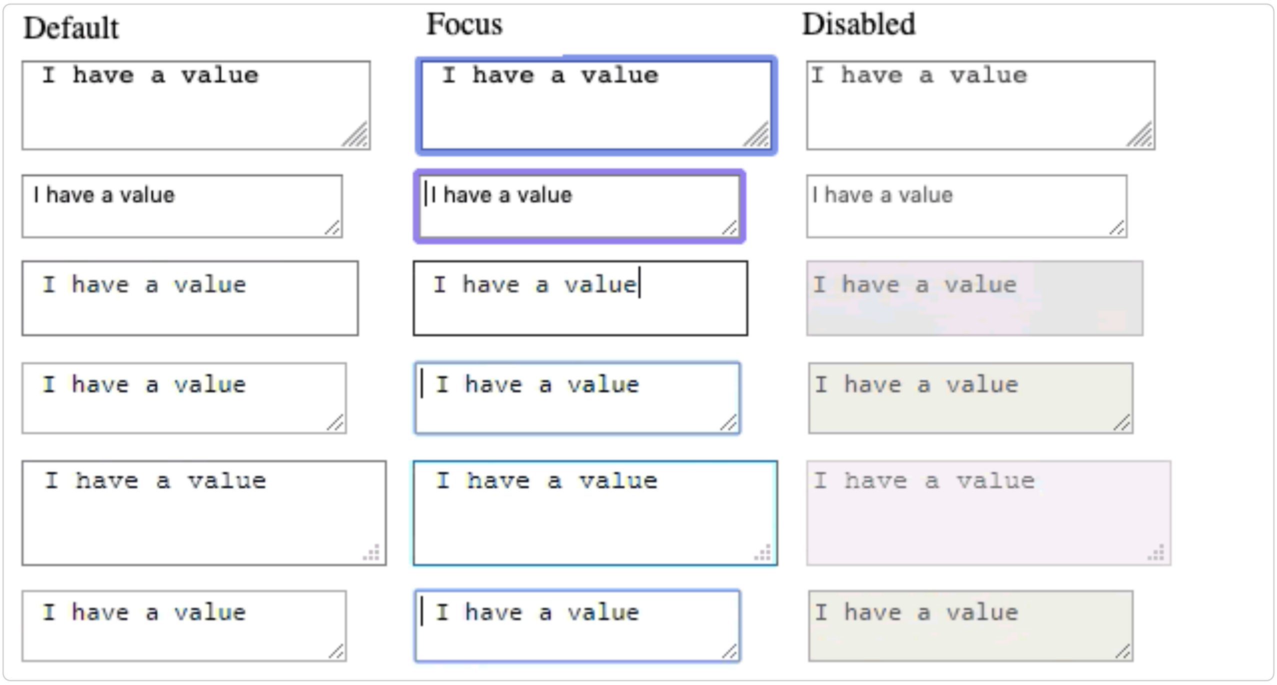
The main difference between a `<textarea>` and a regular single-line text field is that users are allowed to include hard line breaks (i.e. pressing return) that will be included when the data is submitted.

`<textarea>` also takes a closing tag; any default text you want it to contain should be put between the opening and closing tags. In contrast, the `<input>` is a void element with no closing tag — any default value is put inside the `value` attribute.

Note that even though you can put anything inside a `<textarea>` element (including other HTML elements, CSS, and JavaScript), because of its nature, it is all rendered as if it was plain text content. (Using `contenteditable` on non-form controls provides an API for capturing HTML/"rich" content instead of plain text).

Visually, the text entered wraps and the form control is by default resizable. Modern browsers provide a drag handle that you can drag to increase/decrease the size of the text area.

The following screenshots show default, focused, and disabled `<textarea>` elements in Firefox 71 and Safari 13 on macOS and in Edge 18, Yandex 14, Firefox 71, and Chrome 79 on Windows 10.



> Note: You can find a slightly more interesting example of text area usage in the example we put together in the first article of the series (see the source code also).

Controlling multi-line rendering

`<textarea>` accepts three attributes to control its rendering across several lines:

`cols`

Specifies the visible width (columns) of the text control, measured in average character widths. This is effectively the starting width, as it can be changed by resizing the `<textarea>`, and overridden using CSS. The default value if none is specified is 20.

`rows`

Specifies the number of visible text rows for the control. This is effectively the starting height, as it can be changed by resizing the `<textarea>`, and overridden using CSS. The default value if none is specified is 2.

`wrap`

Specifies how the control wraps text. The values are `soft` (the default value), which means the text submitted is not wrapped but the text rendered by the browser is wrapped; `hard` (the `cols` attribute must be specified when using this value), which means both the submitted and rendered texts are wrapped, and `off`, which stops wrapping.

Controlling textarea resizability

The ability to resize a `<textarea>` is controlled with the CSS `resize` property. Its possible values are:

- `both`: The default — allows resizing horizontally and vertically.

- `horizontal` : Allows resizing only horizontally.
- `vertical` : Allows resizing only vertically.
- `none` : Allows no resizing.
- `block` and `inline` : Experimental values that allow resizing in the `block` or `inline` direction only (this varies depending on the directionality of your text; read Handling different text directions if you want to find out more.)

Play with the interactive example at the top of the `resize` reference page for a demonstration of how these work.

Drop-down controls

Drop-down controls are a simple way to let users select from many options without taking up much space in the user interface. HTML has two types of drop-down controls: the select box and the autocomplete box. The interaction is the same in both the types of drop-down controls — after the control is activated, the browser displays a list of values the user can select from.

> Note: You can find examples of all the drop-down box types on GitHub at drop-down-content.html (see it live also).

Select box

A simple select box is created with a `<select>` element with one or more `<option>` elements as its children, each of which specifies one of its possible values.

Basic example

```
<select id="simple" name="simple">
  <option>Banana</option>
  <option selected>Cherry</option>
  <option>Lemon</option>
</select>
```

If required, the default value for the select box can be set using the `selected` attribute on the desired `<option>` element — this option is then preselected when the page loads.

Using optgroup

The `<option>` elements can be nested inside `<optgroup>` elements to create visually associated groups of values:

```
<select id="groups" name="groups">
  <optgroup label="fruits">
    <option>Banana</option>
    <option selected>Cherry</option>
    <option>Lemon</option>
  </optgroup>
  <optgroup label="vegetables">
    <option>Carrot</option>
    <option>Eggplant</option>
    <option>Potato</option>
  </optgroup>
</select>
```

On the `<optgroup>` element, the value of the `label` attribute is displayed before the values of the nested options. The browser usually sets them visually apart from the options (i.e. by being bolded and at a different nesting level) so they are less likely to be confused for actual options.

Using the value attribute

If an `<option>` element has an explicit `value` attribute set on it, that value is sent when the form is submitted with that option selected. If the `value` attribute is omitted, as with the examples above, the content of the `<option>` element is used as the value. So `value` attributes are not needed, but you might find a reason to want to send a shortened or different value to the server than what is visually shown in the select box.

For example:

```
<select id="simple" name="simple">
```

```
  <option value="banana">Big, beautiful yellow banana</option>
  <option value="cherry">Succulent, juicy cherry</option>
  <option value="lemon">Sharp, powerful lemon</option>
</select>
```

By default, the height of the select box is enough to display a single value. The optional `size` attribute provides control over how many options are visible when the select does not have focus.

Multiple choice select box

By default, a select box lets a user select only one value. By adding the `multiple` attribute to the `<select>` element, you can allow users to select several values. Users can select multiple values by using the default mechanism provided by the operating system (e.g., on the desktop, multiple values can be clicked while holding down `Cmd`/`Ctrl` keys).

```
<select id="multi" name="multi" multiple size="2">
  <optgroup label="fruits">
    <option>Banana</option>
    <option selected>Cherry</option>
    <option>Lemon</option>
  </optgroup>
  <optgroup label="vegetables">
    <option>Carrot</option>
    <option>Eggplant</option>
    <option>Potato</option>
  </optgroup>
</select>
```

Note: In the case of multiple choice select boxes, you'll notice that the select box no longer displays the values as drop-down content — instead, all values are displayed at once in a list, with the optional `size` attribute determining the height of the widget.

Note: All browsers that support the `<select>` element also support the `multiple` attribute.

Autocomplete box

You can provide suggested, automatically-completed values for form widgets using the `<datalist>` element with child `<option>` elements to specify the values to display. The `<datalist>` needs to be given an `id`.

The data list is then bound to an `<input>` element (e.g. a `text` or `email` input type) using the `list` attribute, the value of which is the `id` of the data list to bind.

Once a data list is affiliated with a form widget, its options are used to auto-complete text entered by the user; typically, this is presented to the user as a drop-down box listing possible matches for what they've typed into the input.

Basic example

Let's look at an example.

```
<label for="myFruit">What's your favorite fruit?</label>
<input type="text" name="myFruit" id="myFruit" list="mySuggestion" />
<datalist id="mySuggestion">
  <option>Apple</option>
  <option>Banana</option>
  <option>Blackberry</option>
  <option>Blueberry</option>
  <option>Lemon</option>
  <option>Lychee</option>
  <option>Peach</option>
  <option>Pear</option>
</datalist>
```

Datalist support and fallbacks

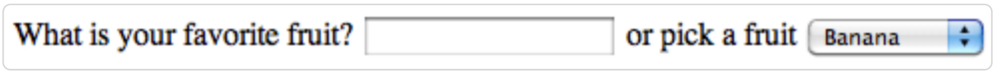
Almost all browsers support datalist, but if you are still supporting older browsers such as IE versions below 10, there is a trick to provide a fallback:

```
<label for="myFruit">What is your favorite fruit? (With fallback)</label>
<input type="text" id="myFruit" name="fruit" list="fruitList" />

<datalist id="fruitList">
  <label for="suggestion">or pick a fruit</label>
  <select id="suggestion" name="altFruit">
    <option>Apple</option>
    <option>Banana</option>
    <option>Blackberry</option>
    <option>Blueberry</option>
    <option>Lemon</option>
    <option>Lychee</option>
    <option>Peach</option>
    <option>Pear</option>
  </select>
</datalist>
```

Browsers that support the `<datalist>` element will ignore all the elements that are not `<option>` elements, with the datalist working as expected. Old browsers that don't support the `<datalist>` element will display the label and the select box.

The following screenshot shows the datalist fallback as rendered in Safari 6:


What is your favorite fruit? or pick a fruit Banana

If you use this fallback, ensure the data for both the `<input>` and the `<select>` are collected server-side.

Less obvious datalist uses

According to the HTML specification , the `list` attribute and the `<datalist>` element can be used with any kind of widget requiring a user input. This leads to some uses of it that might seem a little non-obvious.

For example, in browsers that support `<datalist>` on `range` input types, a small tick mark will be displayed above the range for each datalist `<option>` value. You can see an implementation example of this on the `<input type="range">` reference page.

And browsers that support `<datalist>`s and `<input type="color">` should display a customized palette of colors as the default, while still making the full color palette available.

In this case, different browsers behave differently from case to case, so consider such uses as progressive enhancement, and ensure they degrade gracefully.

Other form features

There are a few other form features that are not as obvious as the ones we have already mentioned, but still useful in some situations, so we thought it would be worth giving them a brief mention.

> Note: You can find the examples from this section on GitHub as other-examples.html (see it live also).

Meters and progress bars

Meters and progress bars are visual representations of numeric values.

Progress

A progress bar represents a value that changes over time up to a maximum value specified by the `max` attribute. Such a bar is created using a `<progress>` element.

```html
<progress max="100" value="75">75/100</progress>
```

This is for implementing anything requiring progress reporting, such as the percentage of total files downloaded, or the number of questions filled in on a questionnaire.

The content inside the `<progress>` element is a fallback for browsers that don't support the element and for screen readers to vocalize it.

Meter

A meter bar represents a fixed value in a range delimited by `max` and `min` values. This value is visually rendered as a bar, and to know how this bar looks, we compare the value to some other set values:

- The `low` and `high` values divide the range into the following three parts:
  - The lower part of the range is between the `min` and `low` values, inclusive.
  - The medium part of the range is between the `low` and `high` values, exclusive.
  - The higher part of the range is between the `high` and `max` values, inclusive.
- The `optimum` value defines the optimum value for the `<meter>` element. In conjunction with the `low` and `high` value, it defines which part of the range is preferred:
  - If the `optimum` value is in the lower part of the range, the lower range is considered to be the preferred part, the medium range is considered to be the average part, and the higher range is considered to be the worst part.
  - If the `optimum` value is in the medium part of the range, the lower range is considered to be an average part, the medium range is considered to be the preferred part, and the higher range is considered to be average as well.
  - If the `optimum` value is in the higher part of the range, the lower range is considered to be the worst part, the medium range is considered to be the average part and the higher range is considered to be the preferred part.

All browsers that implement the `<meter>` element use those values to change the color of the meter bar:

- If the current value is in the preferred part of the range, the bar is green.
- If the current value is in the average part of the range, the bar is yellow.
- If the current value is in the worst part of the range, the bar is red.

Such a bar is created by using the `<meter>` element. This is for implementing any kind of meter; for example, a bar showing the total space used on a disk, which turns red when it starts to get full.

```
<meter min="0" max="100" value="75" low="33" high="66" optimum="50">75</meter>
```

The content inside the `<meter>` element is a fallback for browsers that don't support the element and for assistive technologies to vocalize it.

Support for `<progress>` and `<meter>` is fairly good — there is no support in Internet Explorer, but other browsers support it well.

Test your skills!

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

Summary

As you'll have seen in the last few articles, there are many types of form control. You don't need to remember all of these details at once, and can return to these articles as often as you like to check up on details.

Now that you have a grasp of the HTML behind the different available form controls, we'll take a look at Styling them.

Advanced Topics

- How to build custom form controls

- Sending forms through JavaScript
- Property compatibility table for form widgets