

Styling Forms

Semantics

1 : Use fieldsets to encapsulate similar fields

Generally, forms are made up of inputs inside form tags. When you've got a lot of fields that the user must fill out, it can be easier for both the user and you, the developer, to keep track of input by using fieldsets. The perennial example of this is using fieldsets to separate a billing address and a shipping address.

```
<fieldset>
<span>Billing Address</span><input type="text" />
<span>City</span><input type="text" />
<span>Province</span><input type="text" />
<span>Postal Code</span><input type="text" />
</fieldset>
```

```
<fieldset>
<span>Shipping Address</span><input type="text" />
<span>City</span><input type="text" />
<span>Province</span><input type="text" />
<span>Postal Code</span><input type="text" />
</fieldset>
```

2 : Label Fieldsets with Legends

It hardly makes sense to use a fieldset without giving it a clear name. We can improve the code above by using the legend element to title our fieldsets. The fieldset element has a border by default, and the legend will lay itself over that border.

```
< fieldset >
  < legend >Billing Address</ legend >
  < span >Address</ span >< input type = "text" />
  < span >City</ span >< input type = "text" />
  < span >Province</ span >< input type = "text" />
  < span >Postal Code</ span >< input type = "text" />
</ fieldset >
```

This results in the following:

Billing Address

Address

City

Province

Postal Code

3 : Name your Inputs

If you want to pass form data to a script, each input element needs to have a name; if you are using PHP, these names will become the keys to a super global array, usually \$_POST or \$_GET.

```

1 < fieldset >
2 < span >Billing Address</ span >< input type = "text" name = "billAddress" />
3 < span >City</ span >< input type = "text" name = "billCity" />
4 < span >Province</ span >< input type = "text" name = "billProvince" />
5 < span >Postal Code</ span >< input type = "text" name = "billPC" />
6 </ fieldset >

```

4 : Use the Label Element

Let's continue improving that code; there's nothing inherently wrong with using a span to label the inputs, but the label tag is a born match for inputs.

```

1 < fieldset >
2 < legend >Billing Address</ legend >
3 < label >Address</ label >< input type = "text" name = "billAddress" />
4 < label >City</ label >< input type = "text" name = "billCity" />
5 < label >Province</ label >< input type = "text" name = "billProvince" />
6 < label >Postal Code</ label >< input type = "text" name = "billPC" />
7 </ fieldset >

```

5 : Give Labels the For Attribute

I really like the 'for' attribute; it provides a way to bind a label to an input. The value of 'for' should be the same as the id of the input you want to bind it to.

```

1 < fieldset >
2   < legend >Billing Address</ legend >
3   < label for = "billAddress" >Address</ label >< input type = "text" id = "billAddress" name = "t
4   < label for = "billCity" >City</ label >< input type = "text" id = "billCity" name = "billCity"
5   < label for = "billProvince" >Province</ label >< input type = "text" id = "billProvince" name =
6   < label for = "billPC" >Postal Code</ label >< input type = "text" id = "billPC" name = "billPC
7 </ fieldset >

```

At first, this is one of those things that only seem to affect your code's quality, but they do a special job in the visible content: when the for attribute is defined, the label becomes a 'clickable' area that will focus the input. For example, clicking the label of a text input will focus your cursor in the box; clicking the label of a checkbox will check (or uncheck) the box.

6 : Use optgroup to categorize options

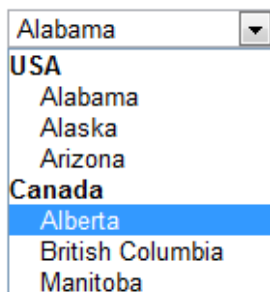
If you have a lot of options in a select, it's usually better to group them into optgroups. <optgroup> is a little-known element that will indent options and give them a title. Note that the label attribute is required.

```

01 < select >
02 < optgroup label = "USA" >
03   < option >Alabama</ option >
04   < option >Alaska</ option >
05   < option >Arizona</ option >
06 </ optgroup >
07 < optgroup label = "Canada" >
08   < option >Alberta</ option >
09   < option >British Columbia</ option >
10   < option >Manitoba</ option >
11 </ optgroup >
12 </ select >

```

This gives us the following results:



7 : Always assign complete attributes

When working with forms particularly, it's tempting to write like this:

```
1 < label for = "live" >Living?</ label >
2 < input name = "live" id = "live" type = "checkbox" checked disabled />
```

Yes, this does what it's supposed to do. No, you shouldn't code like this!

It isn't standards-compliant. Whenever you are adding attributes to an element, don't cut corners.

```
1 < label for = "live" >Living?</ label >
2 < input name = "live" id = "live" type = "checkbox" checked = "checked" disabled = "disabled" />
```

8 : Consider using Buttons instead of Submit Inputs

Generally, `<input type="submit" />` has been the universal submit button. But HTML has a `<button>` element. Why would you use it? Well, it's generally easier to style buttons; also, you can put images within a button, so it really offers more flexibility. You can read more in [these](http://particletree.com/features/rediscovering-the-button-element/)

(<http://particletree.com/features/rediscovering-the-button-element/>)_ [two](http://trevordavis.net/blog/tutorial/input-vs-button/) (<http://trevordavis.net/blog/tutorial/input-vs-button/>)_ articles.

Accessibility

9 : Put tabindexes on your inputs

It's definitely easier to tab through a form than it is to use your mouse . . . however, by default, your user will tab through in the order they are written in the HTML. If this isn't the order you want them to go through the inputs, you can easily add the `tabindex` property to your inputs; `tabindex` takes a number value, and will hop to the input with the next highest value when you hit that tab key.

```
1 < input type = "text" tabindex = "2" />
2 < input type = "text" tabindex = "1" />
3 < input type = "text" tabindex = "3" />
```

10 : Define accesskey when appropriate

The `accesskey` attribute creates a keyboard shortcut that will focus that input: the shortcut is Alt (Option) + the `accesskey` value. Obviously, you wouldn't put an `accesskey` on every input, but it would certainly be useful on, for example, a search box. Don't forget to let users know about the shortcut; often this is done by underlining the letter, as it's usually part of the label.

```

1 < label for = "search" >< span class = "shortcut" >S</ span >earch</ label >
2 < input type = "text" name = "s" id = "search" accesskey = "s" />

```

11 : Use good focusing techniques

You could argue that this point is as much on the side of design as it is accessibility.

It's always nice if a form field (usually a text box, in this case) changes colour when it's focused, but for the visually

impaired, it's almost a requirement if they are to use the form successfully.

To this end, you can use the hover psuedoclass in your CSS; this will work in all common browsers except IE7 and down. You can also use JavaScript for this;

[jQuery has a hover event](http://docs.jquery.com/Events/focus) [_ \(http://docs.jquery.com/Events/focus\)](http://docs.jquery.com/Events/focus)..

```

1 input[type=text]:hover {
2     background-color : #ffff66 ;
3     border-color : #999999 ;
4 }

```

12 : Consider people using Screen Readers

Since forms have the tendency to be so tedious, everyone likes a well-designed form.

But don't let a fancy form ignore screen readers: always make sure your inputs are clearly labeled. If you don't want those labels to show (maybe you are labeling text inputs with values that disappear on focus), you can remove them from the visual presentation (don't use display: none, though;

[there are better ways](http://webaim.org/blog/hiding-content-for-screen-readers/) [_ \(http://webaim.org/blog/hiding-content-for-screen-readers/\)](http://webaim.org/blog/hiding-content-for-screen-readers/)). Also, screen readers generally associate the text directly before an input to be the label for the input. The exceptions to this are radio buttons and checkboxes.

Functionality

13 : Use the right Content Type

In most cases you won't need to put the enctype attribute on your form tag; it will default to "application/x-www-form-urlencoded." However, when you have a file input, which will allow the user to upload the file, you need to use "multipart/form-data."

```

1 < form action = "verify.php" method = "get" enctype = "multipart/form-data" >
2     &tl;label for="avatar">Upload your Avatar : </ label >
3     < input type = "file" name = "avatar" id = "avatar" />
4 </ form >

```

14 : Know when to use Get and when to use Post

A form can send its data by two methods: get and post; you define one in the method attribute on the form tag. What's the difference, and when should you use them? Ignoring what goes on at the server, the main difference is the way the browser sends the information. With get, the form data is send as a query, visible in the url. So this form . . .

```

1 < form action = "you.php" method = "get" >
2   &lt;l;label for="fname">First Name</ label >
3   < input type = "text" name = "fname" id = "fname" value = "Bill" />
4   &lt;l;label for="lname">Last Name</ label >
5   < input type = "text" name = "lname" id = "lname" value = "Gates" />
6 </ form >

```

. . . would result in this URL when submitted: `http://www.example.com/you.php?fname=Bill&lname=Gates`

When you use post, the data is sent in the HTTP request header. That way, it's not visible to the average user. So which should you use when? Post is better for sensitive data (like passwords) and any data that will generally change something (e.g. add a record to the database). Also, post is your only option if you're uploading a file. Get is great for querying the database, and other requests that don't have a lasting affect on anything ("idempotent" the spec calls it). Really, I've just scratched the surface on the differences here: there are [other](http://www.cs.tut.fi/~jkorpela/forms/methods.html) [articles](http://www.cs.tut.fi/~jkorpela/forms/methods.html) [that go into this in-depth.](http://www.diffen.com/difference/Get_vs_Post)

15 : Validate on both the Client and Server

```

function validate(field) {
    var error = field.siblings(".error"),
        value = field.val();

    switch($(field).attr('name')) {
        case "code":
            if(value == ""){
                error.text("Please enter a code").fadeIn();
            } else if (!(/^[A-Za-z]+$/.test(value))) {
                error.text("Please enter only letters").fadeIn();
            } else {
                error.fadeOut();
            }
            break;
        case "number" :
            if(value == "" || !(/^\d{3}$/.test(value))) {

```

Validation is the bane of forms. But it's best to check the input both on the client and on the server; validating in the browser allows you to warn the user of mistakes before they submit the form, which requires one less transaction with the server. However, always be sure to validate on the server as well, for security's sake.

16 : Give your Users Smart Warnings

This goes hand in hand with the previous best practice. Too many times I've submitted a form only to be told "Fields were not filled correctly." Can you spell vague? Once you've determined that your user has made a mistake, let them know as soon and as clearly as possible. Put your error messages close to the bad field, and let your user know what's wrong with their entry. I like using the jQuery's blur() event for this: as soon a user hops to the next box, the previous one is validated.

17 : Consider using AJAX to submit

Many times, submitting a form results in a simple message: "Thank you," "Check your email for confirmation," or "We'll get back to you when we can." When that's the case, what better place to use AJAX? You could just fade out the form, send the data with jQuery or (YOUR FAVOURITE LIBRARY), and fade in your message.

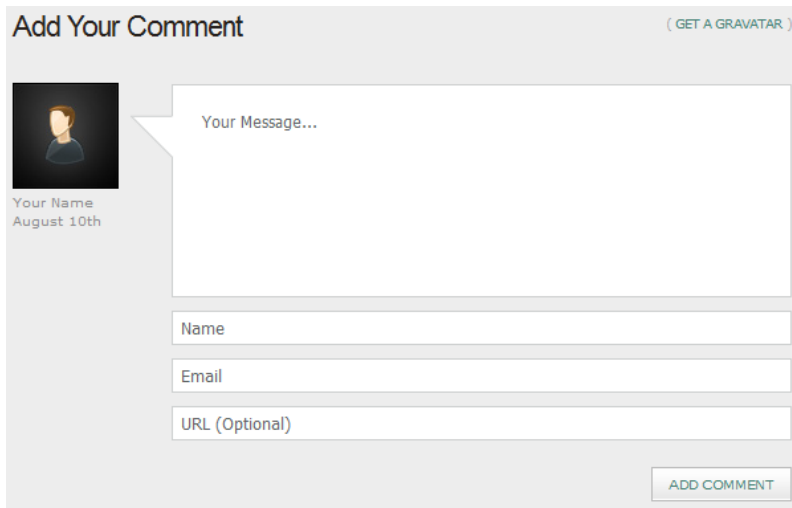
18 : Make sure your form works without Javascript

Maybe this should have gone under accessibility; although the last couple of tips need JavaScript to work at all, make sure your form is completely functional without them. This means a regular form submit, server-side validation, and good errors after a page reload.

Design

19 : Style Forms Consistently

I'm no designer, and I don't pretend to be, but I do know this much: don't fling your form fields carelessly around. Your form should be consistent in its styling. Decide whether your labels will be to the left or right (or perhaps above or below) of the fields, and stick with it. Make all your text inputs and textareas the same width. Space all your fields equally. Keep at least one edge of all boxes aligned. The tuts+ comment forms are great examples of well-styled forms.



20 : Consider using JavaScript to Consistently Style Forms over Different Platforms

With up to ten browsers/operating system combinations, form element consistency is hardly possible . . . without the help of a bit of JavaScript. If you want your forms to look the same on almost every browser, check out the [jqTransform jQuery plugin](http://www.dfc-e.com/metiers/multimedia/opensource/jqtransform/) [\(http://www.dfc-e.com/metiers/multimedia/opensource/jqtransform/\)](http://www.dfc-e.com/metiers/multimedia/opensource/jqtransform/), a plugin aimed directly at this compatibility problem. Simply include it, call it, and adjust the included css file to your taste; it works with IE6+, Safari 2+, Firefox 2+, and Chrome.

21 : Be inspired by others

If you're having trouble coming up with that unique form design for your site, go for a little inspiration! Smashing Magazine has a [great roundup of forms](http://www.smashingmagazine.com/2008/04/17/web-form-design-modern-solutions-and-creative-ideas/) [\(http://www.smashingmagazine.com/2008/04/17/web-form-design-modern-solutions-and-creative-ideas/\)](http://www.smashingmagazine.com/2008/04/17/web-form-design-modern-solutions-and-creative-ideas/), and Smileycat's "Elements of Design" Gallery has a bunch of [Blog Comment Forms](http://www.smileycat.com/design_elements/blog_comment_forms/) [\(http://www.smileycat.com/design_elements/blog_comment_forms/\)](http://www.smileycat.com/design_elements/blog_comment_forms/) worth checking out.

Conclusion

22: Look forward to HTML 5 Forms

HTML 5 has some great features for web forms. Two of the most exciting ones are new types for inputs (like url, email, and date) and the datalist element, which can be used for easy autocomplete. When these and other parts of the spec get implemented, dynamic forms will be much easier!

Website forms can be challenging, but I hope these tips will help you make your forms stand out from the rest.