sitepoint

THE ART & SCIENCE OF CSS

BY CAMERON ADAMS
JINA BOLTON
DAVID JOHNSON
STEVE SMITH
JONATHAN SNOOK

CREATE INSPIRATIONAL STANDARDS-BASED WEB DESIGN

The Art & Science Of CSS

Thank you for downloading this sample chapter of *The Art & Science Of CSS*, published by SitePoint.

This excerpt includes the Summary of Contents, Information about the Author, Editors and SitePoint, Table of Contents, Preface, one chapter from the book (*Forms*), and the index.

We hope you find this information useful in evaluating this book.

Expert Reviewer: Dan Rubin
Expert Reviewer: Jared Christensen
Technical Editor: Andrew Krespanis
Editor: Hilary Reynolds
Cover Design: Alex Walker

Production: *Book*NZ (www.booknz.co.nz)
Managing Editor: Simon Mackie
Technical Director: Kevin Yank
Index Editor: Max McMaster

Printing History
First Edition: March 2007

Published by SitePoint Pty. Ltd.
424 Smith Street Collingwood
VIC Australia 3066.
Web: www.sitepoint.com
Email: business@sitepoint.com

ISBN 978-0-9758419-7-6
Printed and bound in the United States of America

About the Authors

Cameron Adams has been adding to the Internet for over seven years and now runs his own design and development business. He likes to combine the aesthetic with the technological on his weblog, http://www.themaninblue.com/, which contains equal parts of JavaScript, design, and CSS.

Jina Bolton, interactive designer, holds a Bachelor of Fine Arts degree in Computer Arts and Graphic Design from Memphis College of Art. In addition to being featured in *CSS Professional Style* and *Web Designing* magazine, Jina consults for various agencies and organizations, including the World Wide Web Consortium. She enjoys traveling, is learning Italian, and considers herself a sushi enthusiast.

David Johnson is one of those evil .NET developers from Melbourne, Australia. He is the senior developer at Lemonade, http://www.lemonade.com.au/, and his role includes C# programming, database design using SQL Server, and front-end development using XHTML and CSS. He makes up for his evil deeds by being a firm believer in web standards and accessibility, and forcing .NET to abide by these rules. His favourite candy is Sherbies.

Steve Smith lives with his wife, son, and a few miscellaneous animals in South Bend, Indiana, USA. As well as maintaining his personal web site, http://orderedlist.com/, Steve works as an independent web designer, developer, and consultant. He does his best to convince his clients and friends that web standards should be a way of life.

Jonathan Snook has been involved with the Web since ’95, and is lucky to be able to call his hobby a career. He worked in web agencies for over six years and has worked with high-profile clients in government, the private sector, and non-profit organizations. Jonathan Snook currently runs his own web development business from Ottawa, Canada, and continues to write about what he loves on his blog, http://snook.ca/.

About the Expert Reviewers

Dan Rubin is a published author, consultant, and speaker on user interface design, usability, and web standards development. His portfolio and writings can be found on http://superfluousbanter.org/ and http://webgraph.com/.

Jared Christensen is a user experience designer and the proprietor of http://jaredigital.com. He has been drawing and designing since the day he could hold a crayon; he enjoys elegant code, walks in the park, and a well-made sandwich.

About the Technical Editor

Andrew Krespanis moved to web development after tiring of the instant noodles that form the diet of the struggling musician. When he's not diving headfirst into new web technologies, he's tending his bonsai, playing jazz guitar, and occasionally posting to his personal site, http://leftjustified.net/.

About the Technical Director

As Technical Director for SitePoint, Kevin Yank oversees all of its technical publications—books, articles, newsletters, and blogs. He has written over 50 articles for SitePoint, but is best known for his book, *Build Your Own Database Driven Website Using PHP & MySQL*. Kevin lives in Melbourne, Australia, and enjoys performing improvised comedy theater and flying light aircraft.

About SitePoint

SitePoint specializes in publishing fun, practical, and easy-to-understand content for web professionals. Visit http://www.sitepoint.com/ to access our books, newsletters, articles, and community forums.

Table of Contents

Preface

In the early days of CSS, many web designers associated it with boring, square boxes and thin borders. "CSS is *ugly*!" they would cry. It took projects such as CSS Edge[1] and CSS Zen Garden[2] to show the web design world that not only could CSS designs achieve the same aesthetic qualities of their table-based ancestors, but, furthermore, that new and interesting design possibilities were available. Not to mention how much more maintainable the markup is—imagine how very, very happy you'll be if you never again have to stare down the barrel of another day's worth of `table` hacking!

Each chapter of this book will teach you how to style common web site components through practical examples. Along the way, you'll learn many handy techniques for bringing complex designs to life in all modern browsers without needing to resort to messy hacks or superfluous presentational markup. Neither accessibility nor markup quality should be sacrificed to make tricky designs easier to achieve, so the exercises you'll find in this book all use examples of best practice XHTML and CSS. Each chapter progressively builds upon the skills you'll have acquired in previous exercises, giving you a practical toolkit of skills with which to express your own creative ideas.

Who Should Read this Book?

This book is ideal for anyone who wants to gain the practical skills involved in using CSS to make attractive web sites, especially if you're not the type who likes to learn by memorizing a formal specification and then trying to work out which browsers implemented it completely (does *anyone* enjoy reading specifications?). The only knowledge you'll need to have is some familiarity with HTML. This book will give designers the skills they need to implement their ideas, and provides developers with creative inspiration through practical examples.

What's in this Book?

This book contains seven chapters that engage with the fundamental elements of the web page—headings, images, backgrounds, navigation—as well as applied styles such as those used in forms, rounded corners for content boxes, and tables. CSS is inherent in the approaches we'll use in the exercises presented here. These exercises will encourage you to address the questions of art and science in all the design choices you make, as a means to

1 http://meyerweb.com/eric/css/edge/
2 http://csszengarden.com/

create designs that are as beautiful as they are functional. Throughout the book, therefore, considerations of usability are always paramount—both in terms of users of mainstream browsers and those employing assistive technology.

- **Chapter 1: Headings**
  Simultaneously conveying the content and the identity of your site, headings are truly the attention-grabbers of your web page. With only a handful of fonts being available across all browsers, CSS can help you style headings that stand out from the crowd. In this chapter, Cameron Adams will show you how to use image and Flash replacement to gain unlimited creativity in designing headings, while retaining the page's accessibility across all browsers.

- **Chapter 2: Images**
  Images are the windows to your web page's soul. Jina Bolton will teach you stunning ways to display your images as she walks you through a number of attractive examples. You'll learn to create a photo album, as well as to successfully place introductory and in-content images within your pages. The techniques of applying borders, padding, typography, and colors to best present your work are covered in detail in this chapter. You'll also discover effective ways to style those all-important captions.

- **Chapter 3: Backgrounds**
  You've probably already found that CSS has significantly affected the way you use web page backgrounds. Here, David Johnson will explain the properties you'll use on a daily basis to transfer your design visions into light-weight markup and CSS. You'll then work through a case study for a fictional project, in which you'll create a great-looking design that's well supported by all modern browsers. Finally, we'll look to the future to predict the new background capabilities that CSS 3 will bring!

- **Chapter 4: Navigation**
  Navigation is crucial to your users' experience of your web site. Steve Smith will reveal the secrets of successful navigation through a case study involving a fictional design client. You'll build both basic and advanced applications of the main navigation styles in use today, including horizontal, vertical, and tabbed navigation menus, and discover how you can use CSS styling to make your navigation both beautiful and usable.

- **Chapter 5: Forms**
  Forms are the quiet achievers of the web page. In this chapter, Cameron Adams will help you ensure that your forms are available to all users—even those employing assistive technology. You'll learn how to create an attractive form that will allow for

the correct and effective labeling, grouping, layout, and styling of your form elements. Forms needn't be just a tedious necessity—as you'll learn in this chapter, they can be presented in a way that enhances your site's overall impact.

- **Chapter 6: Rounded Corners**
  Those sharp corners on HTML content boxes have been the bane of many a web designer's life for years. But CSS has changed all that, as Steve Smith explains. Flexibility is the key—horizontal, vertical, or even a combination of both forms—to creating rounded corners for your boxes with some straightforward styling. The achievement of rounded corners does hold traps for the unwary, including unsympathetic browsers, but you'll find that taking the few small precautions detailed here will help you to avoid them.

- **Chapter 7: Tables**
  Tables have gained a new lease of life in the CSS era—while they've finally been freed from misuse as a layout element, they retain enormous potential as presenters of data. Jonathan Snook will demonstrate how you can use CSS to create exciting, colorful tables, which will work successfully across browsers. You'll also be invited to envision the future, in which the advent of the wide use of CSS 3 will create even more scope for creative tables.

This Book's Web Site

Located at http://www.sitepoint.com/books/cssdesign1/, the web site supporting this book will give you access to the following facilities.

The Code Archive

The code archive for this book, which can be downloaded from http://www.sitepoint.com/books/cssdesign1/code.php, contains the source code and images for each and every example in this book.

Updates and Errata

The Corrections and Typos page on the book's web site, at http://www.sitepoint.com/books/cssdesign1/errata.php, will always have the latest information about known typographical and code errors, and necessary updates for changes to technologies.

The SitePoint Forums

While we've made every attempt to anticipate any questions you may have, and answer them in this book, there is no way that any publication could cover everything there is to know about designing with CSS. If you have a question about anything in this book, the best place to go for a quick answer is SitePoint's Forums, at http://www.sitepoint.com/forums/—SitePoint's vibrant and knowledgeable community.

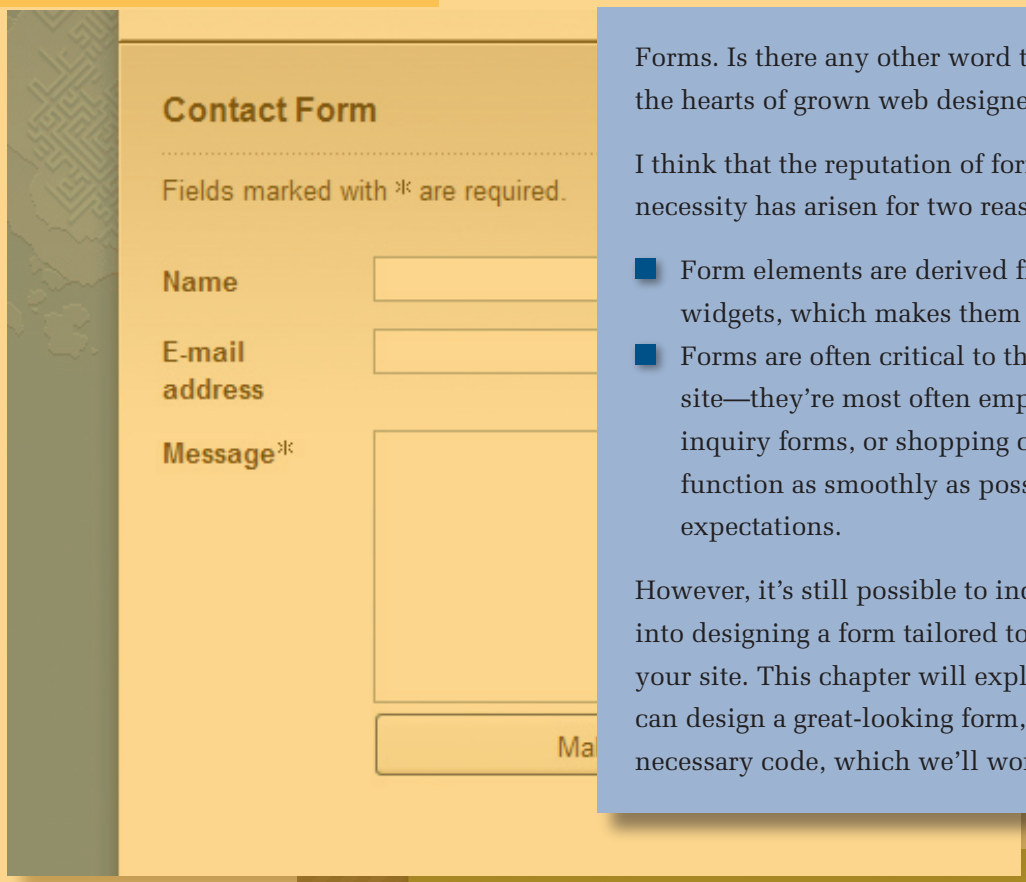
The SitePoint Newsletters

In addition to books like this one, SitePoint offers free email newsletters. The *SitePoint Tech Times* covers the latest news, product releases, trends, tips, and techniques for all technical aspects of web development. The long-running *SitePoint Tribune* is a biweekly digest of the business and moneymaking aspects of the Web. Whether you're a freelance developer looking for tips to score that dream contract, or a marketing major striving to keep abreast of changes to the major search engines, this is the newsletter for you. The *SitePoint Design View* is a monthly compilation of the best in web design. From new CSS layout methods to subtle Photoshop techniques, SitePoint's chief designer shares his years of experience in its pages. Browse the archives or sign up to any of SitePoint's free newsletters at http://www.sitepoint.com/newsletter/.

Your Feedback

If you can't find your answer through the forums, or you wish to contact us for any other reason, the best place to write is books@sitepoint.com. SitePoint has a well-manned email support system set up to track your inquiries, and if the support staff are unable to answer your question, they send it straight to us. Suggestions for improvement as well as notices of any mistakes you may find are especially welcome.

5 Forms

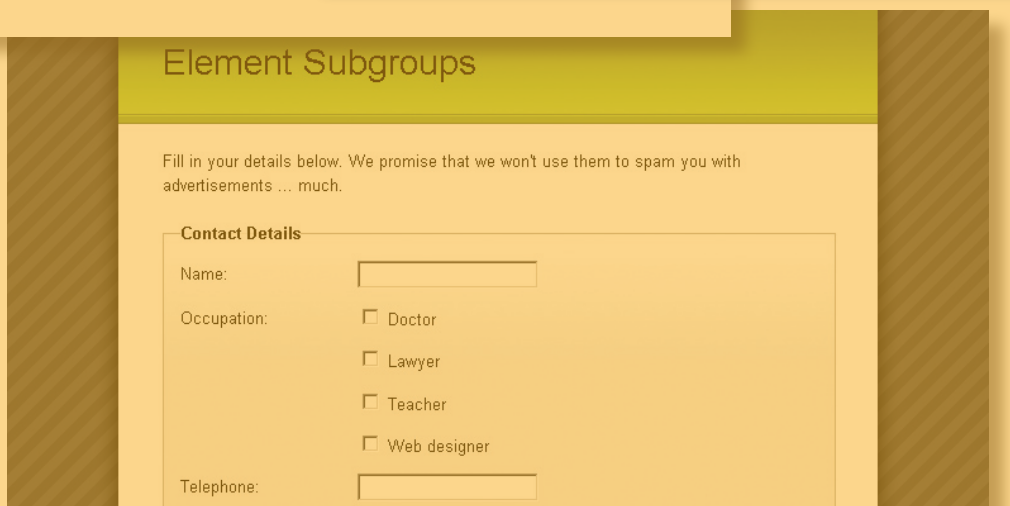


Forms. Is there any other word that strikes as much fear into the hearts of grown web designers?

I think that the reputation of forms as an untamable, ugly necessity has arisen for two reasons:

- Form elements are derived from native operating system widgets, which makes them particularly difficult to style.
- Forms are often critical to the function of a web site—they're most often employed as search boxes, inquiry forms, or shopping cart checkouts—and need to function as smoothly as possible in order to meet user expectations.

However, it's still possible to incorporate both these points into designing a form tailored to the style of the rest of your site. This chapter will explore the ways in which you can design a great-looking form, and provide you with the necessary code, which we'll work work through together.



Accessible Form Markup

Before we can begin to look at form layout, we need to craft some really solid markup that will provide us with a framework to which we can add some style.

Forms represent the one area of your web site where you absolutely *must* commit time and energy to ensure user accessibility. Even though forms represent some of the most complex interactions that can occur on a web page, in many cases these interactions are only represented visually—via the proximity of a form element to its **`label`**, or grouping by borders and background colors. Users of assistive technology such as screen readers may not be able to see these visual clues, so it's vital that you support these users by ensuring accessibility. The key concept behind providing an accessible form is to have descriptive labeling of all its sections and **`input`** elements.

In particular, this means the proper usage of two elements: **`label`** and **`legend`**.

There's also an improperly held belief that the only way you can guarantee that a form displays properly is by using tables. All of the code reproduced here for forms is standards-based, semantic markup, so you've got no excuse for relying on tables now!

Labeling Form Elements

No matter how you style a form element and its **`label`**, it generally conforms to a certain pattern:

- the form element itself
- a text label for the element
- a connection between the element and its textual description

This connection is made either through visual alignment, visual grouping, or some other visual indicator. In Figure 5.1, you can see that the form on the left makes a connection between the field element and its label purely through alignment, whereas the form on the right indicates a more explicit connection via the use of color.

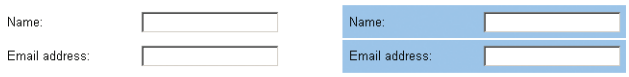


Figure 5.1: Visual connections in forms

When accommodating users of assistive technology in the creation of your forms, there's one main question to consider. How can a user who's unable to see a web page create the connection between a form element and its text label, without the visual cues of proximity or grouping?

The answer is the **label** element. **label** is a special element applied to a form element to allow its textual description to be semantically linked to the element itself, so any assistive technology such as a screenreader can read out that text when it encounters its partner form element.

In order to use a **label**, wrap the textual description in a pair of **labellabel** tags, then add a **for** attribute to the **label**. The value of the **for** attribute should be the **id** of the form element with which you want to create a connection:

```
<label for="firstName">First name</label>
<input id="firstName" name="firstName" type="text" />
```

Now, when a screenreader encounters the **firstName** field, it'll also read out the text "First name" to the user, so he or she will know what to type into that field. The **label** doesn't have to be near the form element and neither of them have to be in any particular order—as long as the **label**'s **for** attribute contains a valid reference, the relationship will be understood. However, having the **label** right before the form element in the source order generally makes the most semantic sense.

A **label** should be applied to any form element that doesn't automatically include descriptive text, such as:

- checkboxes
- radio buttons
- **textarea**s
- text fields
- **select** boxes

Submit buttons and submit images don't require **label** elements, because their descriptions are contained, respectively, in their **value** and **alt** attributes.

Of course, you can easily style the text inside the **label** using CSS, so you can format the **label** text in your forms in the same way as if you were using a **span**, **p**, or **div**, but using a **label** has the benefit of being much more accessible than any of those elements.

Grouping Related Elements

legend goes hand in hand with **fieldset**. In fact, the only element of which a **legend** can be a child *is* a **fieldset**. A **fieldset** groups a series of related form elements. For instance, "street address," "suburb," "state," and "zip code" could all be grouped under "postal address." You could create a **fieldset** that groups all of those elements, and give it an appropriate **legend** to describe that group:

```
<form action="example.php">
  <fieldset>
    <legend>Postal Address</legend>
    <label for="street">Street address</label>
    <input id="street" name="street" type="text" />
    <label for=" suburb">Suburb</label>
    <input id="suburb" name="suburb" type="text" />
    <label for="state">State</label>
    <input id="state" name="state" type="text" />
    <label for="postcode">Postcode</label>
    <input id="postcode" name="postcode" type="text" />
  </fieldset>
</form>
```

Now that **`legend`** is associated with all those form elements inside the **`fieldset`**, when a person using a screenreader focuses on one of the form elements, the screenreader will also read out the **`legend`** text: "Postal Address; Suburb."

The benefit of the screenreader specifying both **`legend`** and **`fieldset`** becomes apparent when you have two groups of elements that are very similar, except for their group type:

```
<form action="example.php">
  <fieldset>
    <legend>Postal Address</legend>
    <label for="street">Street address</label>
    <input id="street" name="street" type="text" />
    <label for=" suburb">Suburb</label>
    <input id="suburb" name="suburb" type="text" />
    <label for="state">State</label>
    <input id="state" name="state" type="text" />
    <label for="postcode">Postcode</label>
    <input id="postcode" name="postcode" type="text" />
  </fieldset>
  <fieldset>
    <legend>Delivery Address</legend>
    <label for="deliveryStreet">Street address</label>
    <input id="deliveryStreet" name="deliveryStreet"
      type="text" />
    <label for="deliverySuburb">Suburb</label>
    <input id="deliverySuburb" name="deliverySuburb"
      type="text" />
    <label for="deliveryState">State</label>
    <input id="deliveryState" name="deliveryState"
      type="text" />
    <label for="deliveryPostcode">Postcode</label>
    <input id="deliveryPostcode" name="deliveryPostcode"
      type="text" />
  </fieldset>
</form>
```

As Figure 5.2 shows, with the **`fieldset`**'s **`legend`** elements in place it's quite easy to determine visually which fields fall under which group, even on an unstyled form.

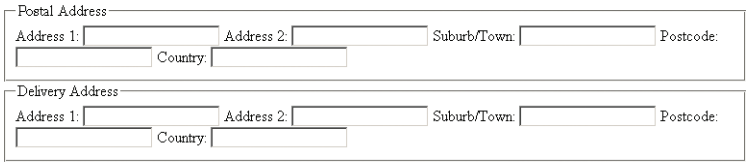


Figure 5.2: Unstyled form using **`fieldset`** and **`legend`** elements for grouping

But, you ask, couldn't the same visual effect be achieved using **`h1`** elements instead of **`legend`** elements?

Yes. However, the point of using **`legend`** is that without proper semantic grouping and labeling, a screenreader user would become confused as to why he or she was required to enter "Address 1" twice. With the **`legend`** included, the user will know that the second "Address 1" actually belongs to another group—the group for the delivery address.

So, by combining **`label`** and **`legend`**, we give visually impaired users the ability to navigate and fill in our forms much more easily. By using this combination as the basic structure for your forms, you'll ensure that not only will they look fantastic, but they'll be accessible as well!

Form Layout

There are several different ways in which you can lay out a form. The method you choose depends upon how long the form is, its purpose, how often it will be used by each person who has to fill it out, and, of course, the general aesthetics of the web page.

It's generally considered most efficient to have one form element per line, with the lines stacked sequentially one on top of the other, as most Western-language web pages are designed to scroll vertically rather than horizontally. This allows users to follow the path to completion easily and focus their attention on entering one piece of data at a time.

For each form element in a left-to-right reading system, it's logical to position the corresponding **`label`** in one of three ways:

- directly above the form element
- in a separate left column, left-aligned
- in a separate left column, right-aligned

Each of these approaches has its own advantages and its own look, so consider these options when you're deciding how to lay out a form for a particular page.

Labels that are positioned directly above a form element have been shown to be processed most quickly by users. The compact grouping between label and element reduces eye movement by allowing the user to observe both simultaneously.[1] However, this type of positioning is rather utilitarian, and isn't the most aesthetically pleasing layout. It also has the disadvantage of occupying the most vertical space of the three layouts, which will make a long form even longer. Generally, top-positioned labels work well for short forms that are familiar to the user, such as the comment form in Figure 5.3.

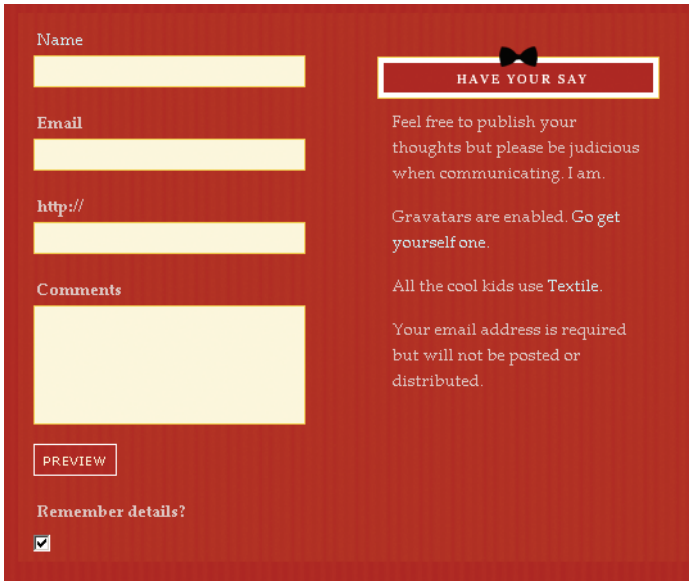


Figure 5.3: Labels positioned above form elements[2]

Labels that are positioned in a column to the left of the elements look much more organized and neat, but the way in which the text in those labels is aligned also affects the usability of the form.

Right-aligning the text creates a much stronger grouping between the label and the element. However, the ragged left edge of the labels can make the form look messy and reduces the ability of users to scan the labels by themselves.[3] In a left-aligned column, the labels instantly become easier to scan, but their grouping with the associated form elements becomes weaker. Users have to spend a little more time correlating the labels with their elements, resulting in slower form completion. An example of left-aligned labels can be seen in Figure 5.4.

1 http://www.uxmatters.com/MT/archives/000107.php
2 http://dressfordialogue.com/thoughts/chris-cornell/
3 http://www.lukew.com/resources/articles/web_forms.html

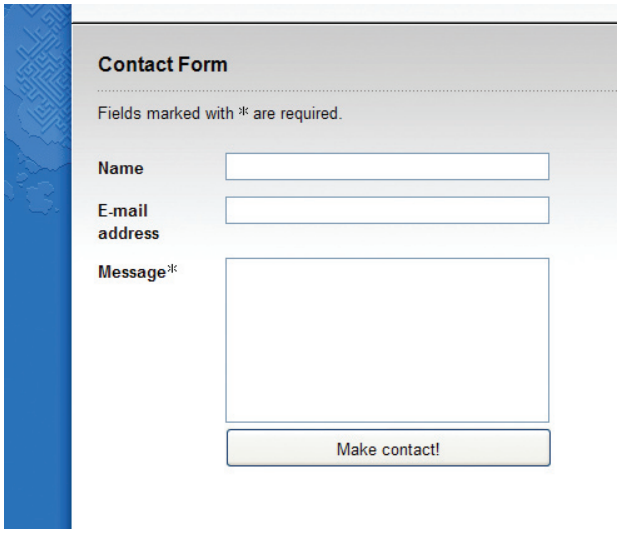


Figure 5.4: Labels positioned in a column and aligned left[4]

The right-aligned column layout shown in Figure 5.5 allows for quicker association between label and element, so again it's more appropriate for forms that will be visited repeatedly by the user. Both layouts have the advantage of occupying a minimal amount of vertical space.

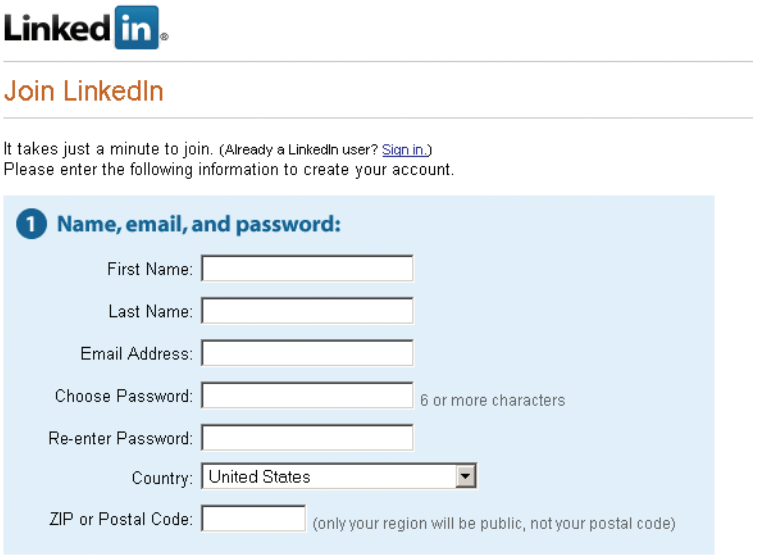


Figure 5.5: Labels positioned in a column and aligned right[5]

4 http://www.themaninblue.com/contact/
5 https://www.linkedin.com/register/

Using the CSS

To create each of these different types of `form` layouts, we'll use identical markup, but with different CSS rules.

In our example, the HTML looks like this:

```
<form action="example.php">
  <fieldset>
    <legend>Contact Details</legend>
    <ol>
      <li>
        <label for="name">Name:</label>
        <input id="name" name="name" class="text" type="text" />
      </li>
      <li>
        <label for="email">Email address:</label>
        <input id="email" name="email" class="text" type="text" />
      </li>
      <li>
        <label for="phone">Telephone:</label>
        <input id="phone" name="phone" class="text" type="text" />
      </li>
    </ol>
  </fieldset>
  <fieldset>
    <legend>Delivery Address</legend>
    <ol>
      <li>
        <label for="address1">Address 1:</label>
        <input id="address1" name="address1" class="text"
            type="text" />
      </li>
      <li>
        <label for="address2">Address 2:</label>
        <input id="address2" name="address2" class="text"
            type="text" />
      </li>
      <li>
        <label for="suburb">Suburb/Town:</label>
        <input id="suburb" name="suburb" class="text"
            type="text" />
      </li>
      <li>
        <label for="postcode">Postcode:</label>
        <input id="postcode" name="postcode"
            class="text textSmall" type="text" />
      </li>
```

```
      <li>
        <label for="country">Country:</label>
        <input id="country" name="country" class="text"
            type="text" />
      </li>
    </ol>


  </fieldset>
  <fieldset class="submit">
    <input class="submit" type="submit"
        value="Begin download" />
  </fieldset>
</form>
```

This HTML uses exactly the same **fieldset-legend-label** structure that we saw earlier in this chapter. However, you should see one glaring addition: inside the **fieldset** elements is an ordered list whose list items wrap around each of the form element/label pairs that we're using.

The reason for this addition? We need some extra markup in order to allow for all of the styling that we'll do to our forms in this chapter. There are just not enough styling hooks in the standard **fieldset-label** structure to allow us to provide robust borders, background colors, and column alignment.

There are a number of superfluous elements that we could add to the form that would grant us the extra styling hooks. We could move the form elements inside their **label** elements and wrap the **label** text in a **span**, or wrap a **div** around each form element/label pair. However, none of those choices would really contribute anything to the markup other than its presence.

The beauty of using an ordered list is that it adds an extra level of semantics to the structure of the form, and also makes the form display quite well in the absence of styles (say, on legacy browsers such as Netscape 4, or even simple mobile devices).

With no CSS applied and without the ordered lists, the rendered markup would appear as in Figure 5.6.

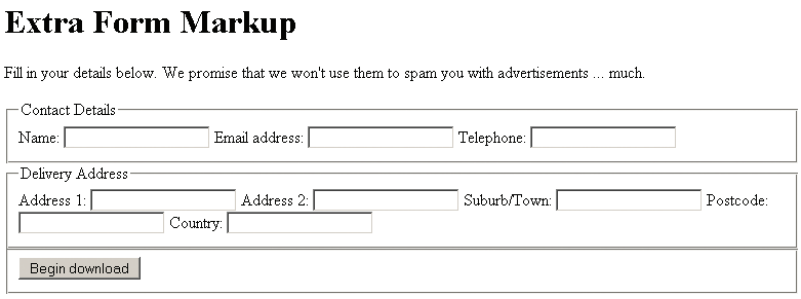


Figure 5.6: Unstyled form without any superfluous markup

Figure 5.7 shows how the unstyled form looks when we include the ordered lists.

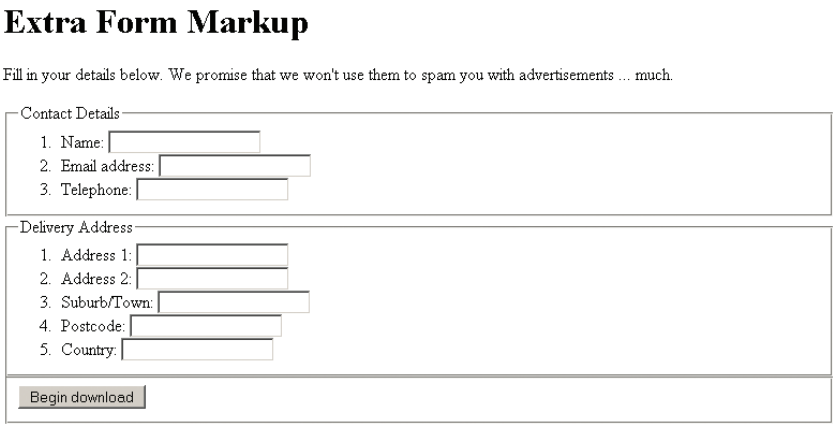


Figure 5.7: Unstyled form that includes an ordered list inside each fieldset

I'm sure you'll agree that the version of the form that includes ordered lists is much easier to follow, and hence fill out.

NOTE *Using Lists in Forms*

If you're vehemently opposed to the inclusion of an ordered list inside your form markup, you can easily substitute it for some other wrapper element; all you need is one extra container around each form element/label pair in order to style your forms any way you want.

Two other HTML oddities that you might have picked up on:

- Each form `input` has a `class` that replicates its `type` attribute, for example `class="text" type="text"`. If you need to style a form element, this is a handy way of accessing it, given that Internet Explorer 6 and earlier don't support CSS attribute selectors (although Internet Explorer 7 does, so you mightn't need to include these extra classes in the near future).
- The form submit button is contained inside its own `fieldset` with `class="submit."` You'll frequently have multiple actions at the end of a form, such as "submit" and "cancel." In such instances, it's quite handy to be able to group these actions, and a `fieldset` does this perfectly. If any styles are applied to normal `fieldset` elements, you'll most often want to have a different style for the `fieldset` surrounding these actions, so the class is necessary to distinguish our actions `fieldset`. The `fieldset` and the input inside it both have the same class name because the term "submit" makes sense for both of them, but it's easy to distinguish them in the CSS by preceding the class selector with an element selector, as we'll see below.

Applying General Form Styling

There are a number of styles which we'll apply to our forms, irrespective of which layout we choose. These styles revolve mainly around the inclusion of whitespace to help separate form elements and **fieldset** elements:

```
fieldset {
  margin: 1.5em 0 0 0;
  padding: 0;
}
legend {
  margin-left: 1em;
  color: #000000;
  font-weight: bold;
}
fieldset ol {
  padding: 1em 1em 0 1em;
  list-style: none;
}
fieldset li {
  padding-bottom: 1em;
}
fieldset.submit {
  border-style: none;
}
```

The **margin** on the **fieldset** helps to separate each **fieldset** group from the others. All internal **padding** is removed from the **fieldset** now, because later on it'll cause problems when we begin floating elements and giving them a **width**. Since **padding** isn't included in the **width**, it can break the dimensions of your form if you have a **width** of **100%** and some **padding**. Removing **padding** also helps to sort out inconsistencies between browsers as to the default internal spacing on the **fieldset**.

To help define a visual hierarchy that clearly shows each **label** inside the **fieldset** grouped under the **legend**, we give our **legend** elements a **font-weight** of **bold**. We also have to replace the spacing that was removed from the **padding** on the **fieldset**, so we give the **legend** a **margin-left** of **1em**.

In order to turn off the natural numbering that would appear for the ordered list, we set **list-style** to **none** on the **ol**, and thus remove any of the bullet formatting that normally exists in such a list. Then, to recreate the internal spacing which we removed from the **fieldset**, we give the ordered list some **padding**. No **padding** is put on the bottom of the list, because this will be taken up by the **padding** of the last list item.

To separate each form element/label pair from each other pair, we give the containing list item a **padding-bottom** of **1em**.

Finally, to remove the appearance of the submit button as a form element group, we need to take the borders off its surrounding **`fieldset`**. This step is achieved by targeting it using the **`fieldset.submit`** selector and setting the **`border-style`** to **`none`**.

After applying all of this markup and adding some general page layout styles, we end up with Figure 5.8—a form that's beginning to take shape, but is still a bit messy.

Figure 5.8: Form with general styling applied, but no layout styles

Now we can go ahead and add in some layout styles!

Using Top-positioned Text Labels

Positioning labels at the top of their form elements is probably the easiest layout to achieve, as we only need to tell the **`label`** to take up the entire width of its parent element.

As our form elements/labels are inside ordered list items (which are block elements), each pair will naturally fall onto a new line, as you can see from Figure 5.9. All we have to do is get the form elements and labels onto different lines.

This exercise is easily completed by turning the **`label`** elements into block elements, so that they'll occupy an entire line:

```
label {
  display: block;
}
```

It's a simple change, but one which makes the form much neater, as shown in Figure 5.9.

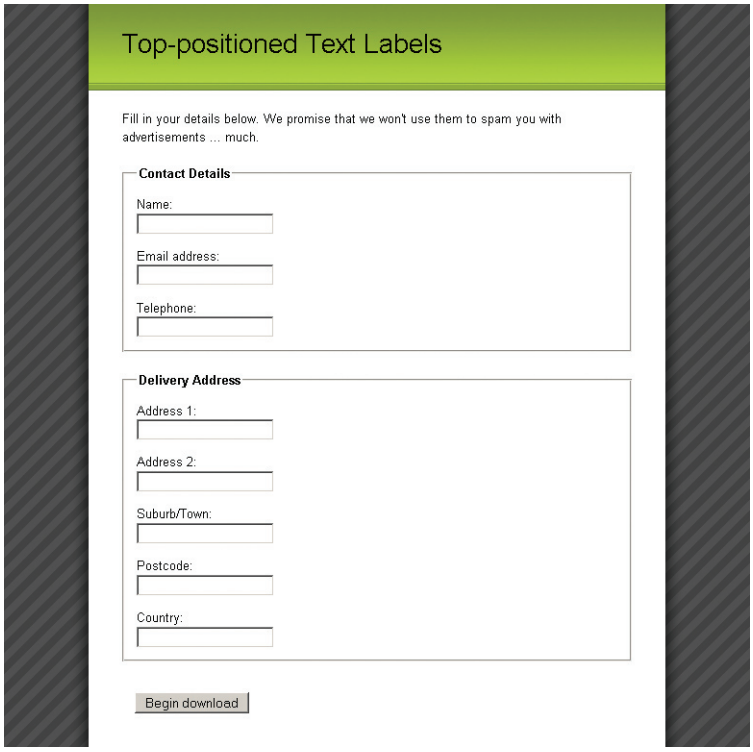


Figure 5.9: Example form with text labels positioned at the top of each form element

Left-aligning Text Labels

When we create a column of text labels to the left of the form elements, we'll have to do a little bit more work than just to position them at the top. Once we begin floating elements, all hell breaks loose!

In order to position the labels next to the form elements, we **`float`** the **`label`** elements to the left and give them an explicit **`width`**:

```
label {
  float: left;
  width: 10em;
  margin-right: 1em;
}
```

We also apply a little bit of **`margin-right`** to each **`label`**, so that the text of the **`label`** can never push right up next to the form element. We must define an explicit **`width`** on the floated element so that all the form elements will line up in a neat vertical column. The exact width we apply will depend upon the length of the form labels. If possible, the longest form label should be accommodated without wrapping, but there shouldn't be such a large gap that the smallest label looks like it's unconnected to its form element. In

the latter scenario, it is okay to have a **`label width`** that is smaller than the longest **`label`**, because the text will wrap naturally anyway, as you can see in Figure 5.10.

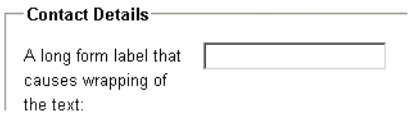


Figure 5.10: Text in floated **`label`** wraps automatically

Once we float the **`label`**, however, we run into a problem with its containing list item—the list item will not expand to match the height of the floated element. This problem is highly visible in Figure 5.11, where we've applied a **`background-color`** to the list item.

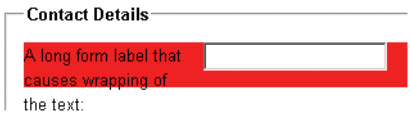


Figure 5.11: **`li`** containing floated **`label`** does not expand to match **`label`** height

One markup-free solution to ensuring a parent contains any of its floated children is to also float the parent, so that's what we'll do:

left-aligned-labels.css (excerpt)

```
fieldset li {
  float: left;
  clear: left;
  width: 100%;
  padding-bottom: 1em;
}
```

If the list item is floated, it'll contain all of its floated children, but its **`width`** must then be set to **`100%`**, because floated elements try to contract to the smallest width possible. Setting the **`width`** of the list item to **`100%`** means that it'll still behave as if it were an unfloated block element. We also throw a **`clear :left`** property declaration in there to make sure that we won't find any unwanted floating of list items around **`form`** elements. **`clear: left`** means that the list item will always appear beneath any prior left-floated elements instead of beside them.

However, once we float the list item, we find the same unwanted behavior on the **`fieldset`**—it won't expand to encompass the floated list items. So, we have to float the **`fieldset`**. This is

the main reason that we removed the **padding** from **fieldset** earlier—when we set its **width** to **100%**, any **padding** will throw out our dimensions:

left-aligned-labels.css (excerpt)

```
fieldset {
  float: left;
  clear: left;
  width: 100%;
  margin: 0 0 1.5em 0;
  padding: 0;
}
```

Where will this float madness end? Remain calm. It ends right here, with the submit **fieldset**. Since it's the last **fieldset** in the form, and because it doesn't need as much special CSS styling as the other **fieldset**s, we can turn off that floating behavior for good:

left-aligned-labels.css (excerpt)

```
fieldset.submit {
  float: none;
  width: auto;
  border: 0 none #FFF;
  padding-left: 12em;
}
```

By turning off floating and setting the **width** back to **auto**, the final submit **fieldset** becomes a normal block element that clears all the other floats. This means the form will grow to encompass all the **fieldset** elements, and we're back in the normal flow of the document.

None of the elements in the submit **fieldset** are floated, but we want the button to line up with all of the other form elements. To achieve this outcome, we apply **padding** to the actual **fieldset** itself, and this action pushes the submit button across to line up with all the text fields. It's best to have the button line up with the form elements, because it forms a direct linear path that the user's eye can follow when he or she is completing the form.

After all that floating, we now have Figure 5.12—a form with a column for the form labels and a column for the form elements.

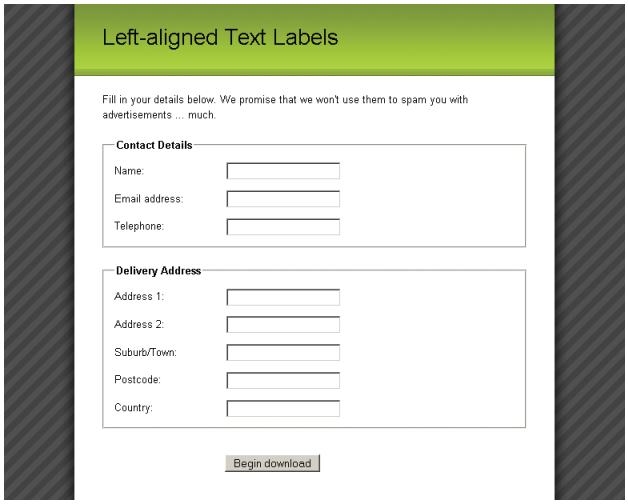


Figure 5.12: Example form with **`label`** elements organized in left-aligned column

Right-aligning Text Labels

With all that difficult floating safely out of the way, aligning the **`input`** labels to the right is a breeze; simply set the text alignment on the **`label`** elements to achieve a form that looks like Figure 5.13:

right-aligned-labels.css (excerpt)

```
label {
  float: left;
  width: 10em;
  margin-right: 1em;
  text-align: right;
}
```

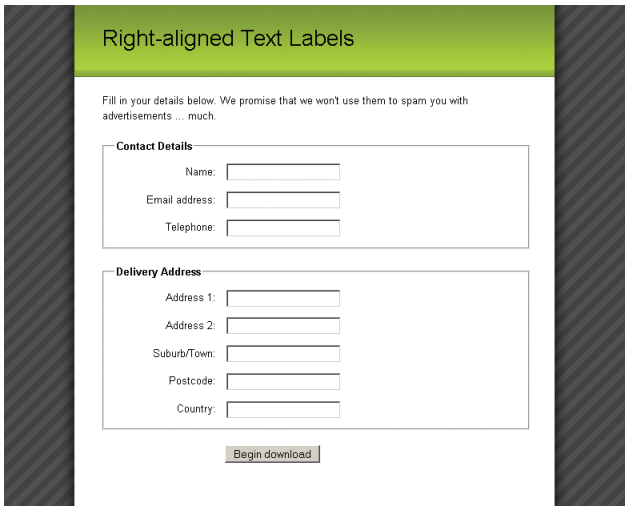


Figure 5.13: Example form with **`label`** elements organized in right-aligned column

And we're done! Now you can take your pick of whichever form layout best fits your pages, all by changing a little CSS!

Applying `fieldset` and `legend` Styles

It's actually fairly rare to see a **`fieldset`** displayed in the default browser style. For some reason people just don't like the look of them, and I must admit those borders and **`legend`** elements don't fit into a lot of page designs. **`legend`** elements are one of the trickiest HTML elements to style, but you can use a number of tricks to tame them, and there are some great ways to differentiate **`fieldset`** elements using CSS.

Providing a background color for your **`fieldset`** elements helps to differentiate **`form`** content from normal content and focuses the user's attention on the **`form`** fields themselves. However, it's not as simple as just specifying a **`background-color`**.

Resolving Internet Explorer's Legends Issues

In a totally unexpected turn of events (yeah, right!) Internet Explorer handles legends differently from other browsers. From experimentation, it seems that Internet Explorer treats **`legend`** elements as if they're *inside* the **`fieldset`**, while other browsers treat them as if they're *outside* the **`fieldset`**. I'm not saying that any browser's wrong, but we have to circumvent these differences somehow, and creating a separate IE style sheet seems to be the best solution.

If you put a **`background-color`** on a **`fieldset`** with a **`legend`**, as in Figure 5.14, you can see the problem all too clearly.

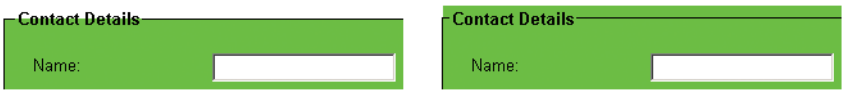


Figure 5.14: Browser rendering of **`fieldset`** elements with background color

The **`fieldset`** on the left shows how most browsers render a **`legend`** and **`fieldset`** with a background color. The **`fieldset`** on the right shows how Internet Explorer renders it—the **`background-color`** of the **`fieldset`** appears to extend beyond its border, stretching to fit the height of the **`legend`**.

The way to avoid this problem is to accomodate Internet Explorer browsers with a separate style sheet that uses conditional comments:

```
<!--[if lte IE 7]>
  <style type="text/css" media="all">
    @import “css/fieldset-styling-ie.css";
  </style>
<![endif]-->
```

This statement includes a style sheet for Internet Explorer 7 and earlier, as these are the versions that currently display this deviant behavior. Any other browsers will ignore it. We could use a style sheet that applies to any version of Internet Explorer—including those released in the future—but the **legend** display difference may be corrected by then, so it's safest just to apply it to the versions we know for the present.

Inside that style sheet we use relative positioning on the **legend** to move it up to align with the top of the **fieldset**:

```
legend {
  position: relative;
  left: -7px;
  top: -0.75em;
}
fieldset ol {
  padding-top: 0.25em;
}
```

In this case, the value we've given the **legend**'s **top**—**0.75em**—just happens to be the right value to get the **legend** to align with the **fieldset**. It may vary depending on other styles we might apply to the legend (such as margin and padding). This is quite a robust solution—we've used relative units, so if users change the text size in their browsers, the position of the **legend** will shift accordingly and still line up.

In addition to moving the top of the **legend**, we move it 7px to the left by applying a **left** value of **-7px**. This step counters an Internet Explorer quirk—IE always shifts legends to the right by 7px (regardless of text size), so we need to negate that shift to get the **legend** and the **label** elements lining up neatly.

Because we're moving the legend up relatively, it will create more space below the **legend**. To counteract this space, we reduce the padding at the top of the ordered list by an equivalent amount, changing it from the original value of **1em** to **0.25em**.

The last Internet Explorer fix is to relatively position the **fieldset** itself:

```
fieldset {
  position: relative;
}
```

Without this rule, Internet Explorer produces some weird visual effects around the **`legend`**. How weird? You can see exactly how weird in Figure 5.15.

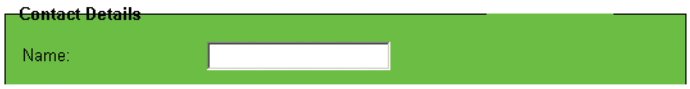


Figure 5.15: Visual aberrations in Internet Explorer

We really need to avoid the IE aberrations we've seen, but we're almost there—now we'll just set the **`position`** of the **`fieldset`** to **`relative`** to restore everything to normal.

Styling Legends and Fieldsets

In all browsers, **`legend`**s will have some **`padding`** by default. The amount of **`padding`** varies between browsers, so to have the **`legend`** lining up nicely with our **`label`**s we'll eliminate the **`padding`** in our main style sheet:

fieldset-background-color.css (excerpt)

```
legend {
  margin-left: 1em;
  padding: 0;
  color: #000;
  font-weight: bold;
}
```

The default **`border`** for **`fieldset`** elements is normally an inset border—which doesn't match some sites—so here we're going to make it a flat, 1px border. In addition, we'll add in a background color that will make the **`fieldset`** elements stand out from the normal page background, marking them as special areas:

fieldset-background-color.css (excerpt)

```
fieldset {
  float: left;
  clear: both;
  width: 100%;
  margin: 0 0 1.5em 0;
  padding: 0;
  border: 1px solid #BFBAB0;
  background-color: #F2EFE9;
}
```

Generally speaking, we don't want any borders or background color behind the submit **`fieldset`**, so it's quite easy to turn those off:

fieldset-background-color.css (excerpt)

```
fieldset.submit {
  float: none;
  width: auto;
  border-style: none;
  padding-left: 12em;
  background-color: transparent;
}
```

Now we've got **fieldset** elements with a background color and a **legend** that lines up neatly with all the other form elements, as in Figure 5.16.

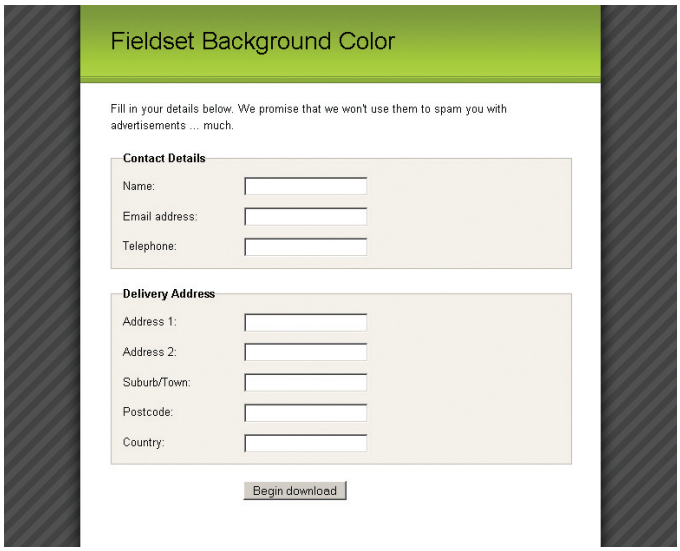


Figure 5.16: **fieldset** elements with **background-color** set and adjustments made to **legend**

The cut-off of color behind the **legend** can sometimes look a bit abrupt, as you can see in the magnified view of the **legend** shown in Figure 5.17.

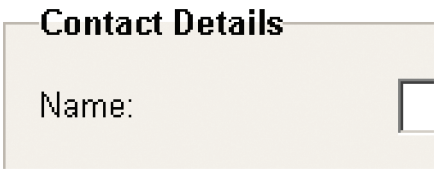


Figure 5.17: Magnification of **legend**—cut-off of background color is apparent

This cut-off will become more pronounced if we use a **fieldset** background color that has more contrast with the normal page background color. If you want to counteract this effect, it's possible to put a gradient background image into the **fieldset** that smoothly changes the color from the page background color (white) to your chosen **fieldset** background color:

fieldset-background-image.css (excerpt)

```
fieldset {
  float: left;
  clear: both;
  width: 100%;
  margin: 0 0 1.5em 0;
  padding: 0;
  border: 1px solid #BFBAB0;
  background-color: #F2EFE9;
  background-image: url(images/fieldset_gradient.jpg);
  background-repeat: repeat-x;
}
```

That **background-image** rule will also be applied to our submit **fieldset**, so to keep a clean, transparent background, we'll also have to cancel the **background-image** on the submit **fieldset**:

fieldset-background-image.css (excerpt)

```
fieldset.submit {
  float: none;
  width: auto;
  border-style: none;
  padding-left: 12em;
  background-color: transparent;
  background-image: none;
}
```

See Figure 5.18—the form looks a lot smoother, no?

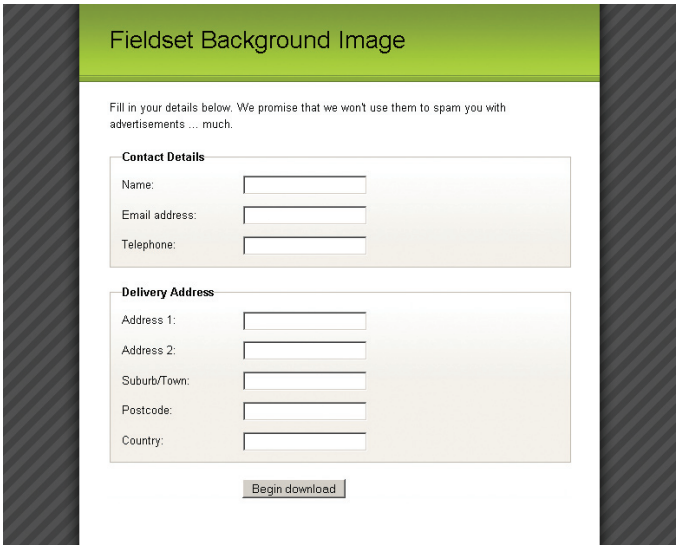


Figure 5.18: **fieldset** elements with background color and gradient images applied

Changing the Default Fieldset Layout

Although **fieldset** and **legend** elements are the most accessible means of marking up form groups, in the past a lot of people haven't used them because they don't like the default styling that browsers impose on these elements—the border around the **fieldset**, the **legend** intersecting the edge of the box. But it *is* possible to change this default layout and make your forms a little less boxy.

Our first step is to push the **fieldset** elements together, eliminating the whitespace between them. To do this, we could make the **margin** on the bottom of the **fieldset** elements zero, but that actually ends up looking like Figure 5.19.

Figure 5.19: **legend** adding extra height so **fieldset** elements cannot touch

The **legend** at the top of the **fieldset** elements prevents the two **fieldset** elements from joining.To circumvent this problem we can use some negative **margin** on the bottom of each **fieldset**. This will "pull" up the lower **fieldset** so that it overlaps the upper **fieldset**, making it look like they're touching.

To prevent the bottom **fieldset** from overlapping any form elements, we should also add a bit of padding to the bottom of the **fieldset** elements so that they've got some space to move into:

```
fieldset {
  float: left;
  clear: both;
  width: 100%;
  margin: 0 0 -1em 0;
  padding: 0 0 1em 0;
  border: 1px solid #BFBAB0;
  background-color: #F2EFE9;
}
```

Moving the **fieldset**s up by **1em** is enough to cover the gap between them, and the **bottom-padding** of **1em** counteracts the movement, making sure no **form** elements disappear beneath **fieldset** elements.

A couple of visual tweaks are necessary when removing the whitespace. Without contact

between the **fieldset** background color and the normal page background color, we no longer need the gradient background image, so this has been left out.

The **border-style** has also been changed—we're removing all borders, then replacing only the top border:

```
fieldset {
  float: left;
  clear: both;
  width: 100%;
  margin: 0 0 -1em 0;
  padding: 0 0 1em 0;
  border-style: none;
  border-top: 1px solid #BFBAB0;
  background-color: #F2EFE9;
}
```

With all the **fieldset** elements being joined together, the extra borders on the left and right make the form look cluttered. With just a top border, we've created a much cleaner look, as shown in Figure 5.20.

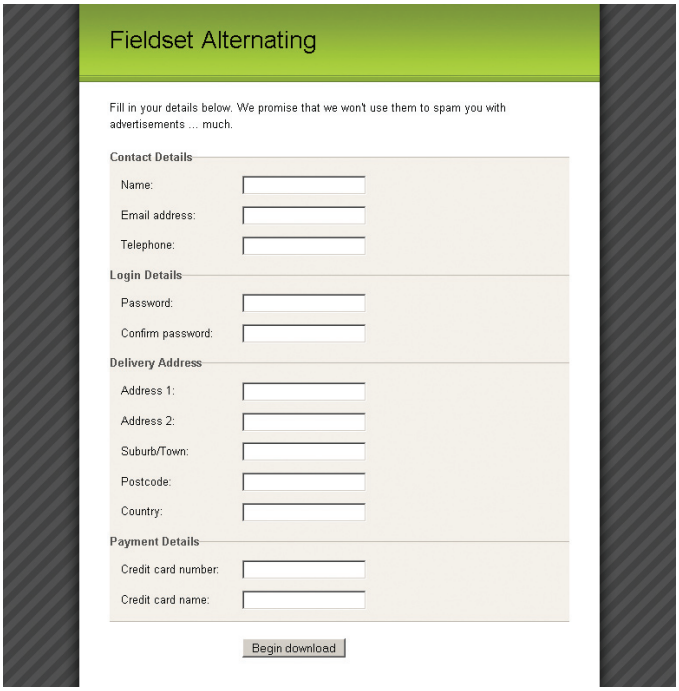


Figure 5.20: Joined **fieldset** elements

The other side effect of joining the **fieldset** elements together is that the **legend** now looks out of place, balancing in between either **fieldset**. The way to solve this problem is to bring the **legend** fully within the boundaries of its **fieldset**.

Instinctively, you might use relative or absolute positioning on the **legend** to move it down into the **fieldset**. However, Firefox resists any attempt to reposition the **legend**—it just doesn't move.

Unfortunately, the only way around this issue is to add a tiny bit more markup to our form. By inserting a superfluous **span** into each of our **legend** elements, Firefox allows us to style this and move the text down into the **fieldset**:

fieldset-alternating.html (excerpt)

```
<legend>
  <span>Contact Details</span>
</legend>
```

That **span** can be positioned absolutely and moved down into the **fieldset** using **margin-top**. While we're at it, let's also increase the **font-size** of the **legend** text, to give it a bit more prominence:

fieldset-alternating.css (excerpt)

```
legend span {
  position: absolute;
  margin-top: 0.5em;
  font-size: 135%;
}
```

There's actually an esoteric bug in some point releases of Firefox (Firefox 1.5.0.6 on Windows XP, but not OSX, from what I've seen) that makes the absolutely positioned **span** elements behave as if they were all positioned at the top of the form element. Giving the **legend** elements a **position** of **relative** doesn't seem to affect the **span** elements, so we actually need to relatively position each of the **fieldset** elements and give the **span** elements some explicit coordinates to sidestep this bug:

fieldset-alternating.css (excerpt)

```
fieldset {
  position: relative;
  float: left;
  clear: both;
  width: 100%;
  margin: 0 0 -1em 0;
  padding: 0 0 1em 0;
  border-style: none;
  border-top: 1px solid #BFBAB0;
  background-color: #F2EFE9;
}
```

```
legend span {
  position: absolute;
  left: 0.74em;
  top: 0;
  margin-top: 0.5em;
  font-size: 135%;
}
```

The `0.74em` value of `left` actually matches the `1em` `padding` we gave to the ordered list, due to the fact that the `span` has a larger `font-size`.

Because we're now specifying a `left` ordinate for the `span`, we also have to take the `margin-left` off its parent `legend`, so that we don't get a doubling of the spacing. Simply omit the `margin` rule that we used previously:

fieldset-alternating.css (excerpt)

```
legend {
  padding: 0;
  color: #545351;
  font-weight: bold;
}
```

That bug's now squashed!

As we're moving the `legend` down into the `fieldset`, we need to make sure that the `legend` won't overlap any of the form elements, so let's add a bit more `padding` to the top of our ordered list:

fieldset-alternating.css (excerpt)

```
fieldset ol {
  padding: 3.5em 1em 0 1em;
  list-style: none;
}
```

Don't forget to change the matching value inside our Internet Explorer-only style sheet:

fieldset-alternating-ie.css (excerpt)

```
legend span {
  margin-top: 1.25em;
}
fieldset ol {
  padding-top: 3.25em;
}
```

Internet Explorer has slightly different spacing on the **legend** element's **span**, so let's tweak the **margin-top** value for that as well.

After all these changes, there's one **fieldset** that looks a little out-of-place: the submit **fieldset**. Because the submit **fieldset** doesn't have a **legend**, the submit button will be moved up too high, so we need to push it down a bit. This is done most easily by adding some **padding** to the top of this **fieldset** only. Also, because the submit **fieldset** will overlap the **fieldset** above it, we need to provide a solid **background-color** for the submit **fieldset**, otherwise the previous **fieldset**'s **background-color** will shrow through. This means changing the **background-color** value from **transparent** to whatever your normal page **background-color** is:

fieldset-alternating.css (excerpt)

```
fieldset.submit {
  float: none;
  width: auto;
  padding-top: 1.5em;
  padding-left: 12em;
  background-color: #FFFFFF;
}
```

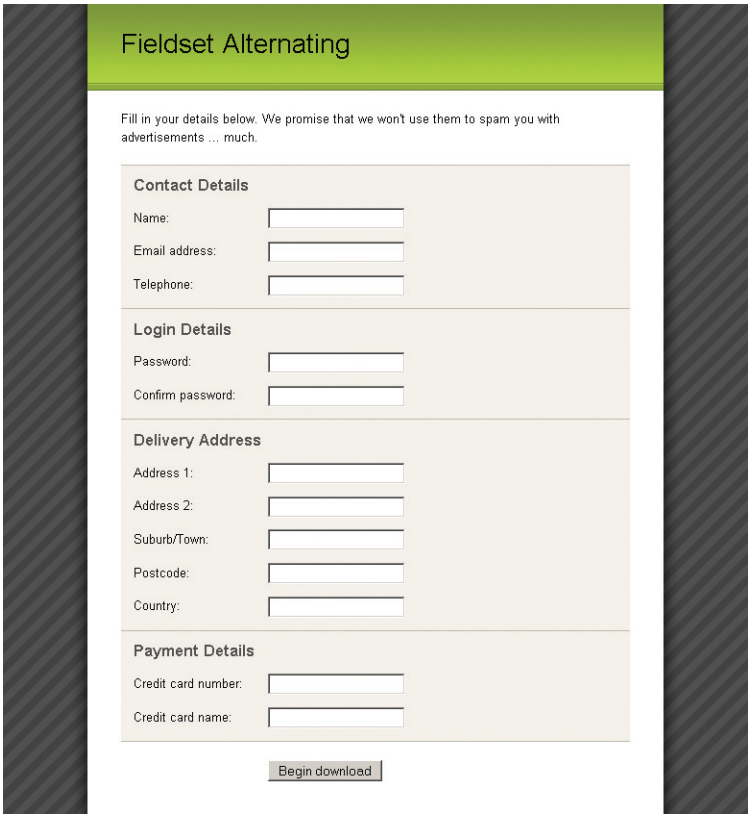


Figure 5.21: All **fieldset** elements joined and **legend** elements moved inside boxes

Previously, we also removed borders from the submit **fieldset**, but for this adjoining layout we need the submit **fieldset** to retain the top border that's applied to all **fieldset** elements. We'll just let that rule cascade into the submit **fieldset** without interference.

Once we've implemented all those changes, the layout of the form is complete. The form appears as shown in Figure 5.21, but it requires some slight aesthetic tweaks.

Because we've pushed all the **fieldset** elements together, they tend to run into one another visually. Better distinction can be

created between each **fieldset** by subtle alternation of the **background-color** elements in odd and even **fieldset** elements. The only cross-browser method for achieving this is to add in a new class for every second **fieldset**. This allows us to use a CSS selector to give those **fieldset** elements a different **background-color**. I normally use a **class** of **alt**, but you can use whatever you think is logical:

```
<fieldset>
…
</fieldset>
<fieldset class="alt">
…
</fieldset>
<fieldset>
…
</fieldset>
<fieldset class="alt">
…
</fieldset>
…
```

Then all you have to do is think of a different **background-color**:

fieldset-alternating.css (excerpt)

```
fieldset.alt {
  background-color: #E6E3DD;
}
```

And our final **form** with alternating **fieldset** elements looks like Figure 5.22!

Grouping Radio Buttons and Checkboxes

There are two types of form elements that are likely to be part of their own subgroup. These are checkboxes and radio buttons, both of which can be used to offer users multiple choices when responding to a given question on a form.

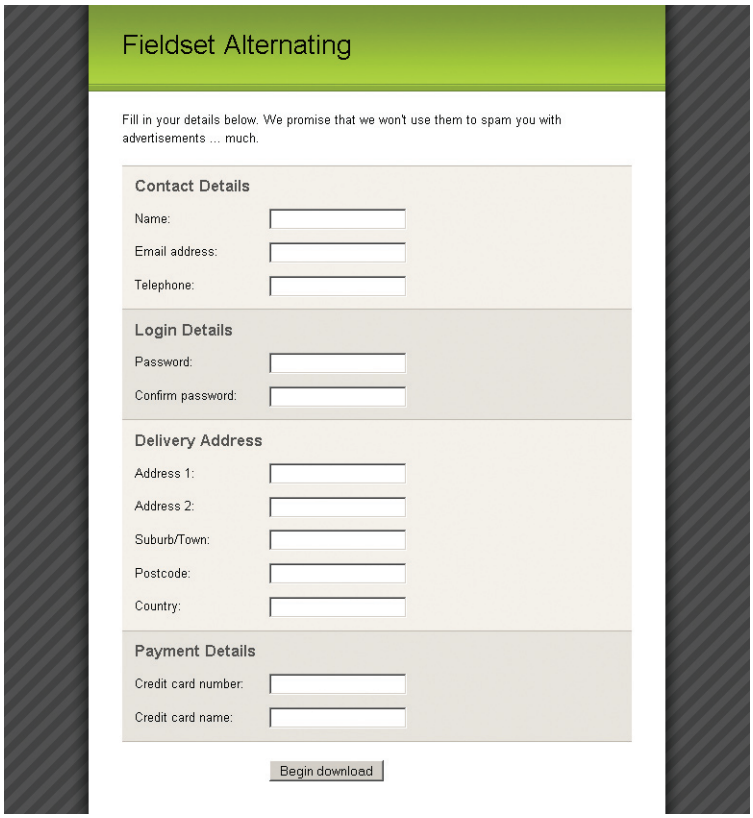


Figure 5.22: Alternating-color **fieldset** elements

The way in which these form elements are laid out is slightly different to text fields, **`select`** boxes or **`textareas`**. As they are part of their own subgroup, they should be included in a nested **`fieldset`** inside the main **`fieldset`**. Using our **`background-image form`** as a starting point, we can add some grouped elements inside the **`fieldset`**:

element-subgroups.html (excerpt)

```
<fieldset>
  <legend>Contact Details</legend>
  <ol>
    <li>
      <fieldset>
        <legend>Occupation:</legend>
        <ol>
          <li>
            <input id="occupation1" name="occupation1"
                class="checkbox" type="checkbox" value="1" />
            <label for="occupation1">Doctor</label>
          </li>
          <li>
            <input id="occupation2" name="occupation2"
                class="checkbox" type="checkbox" value="1" />
            <label for="occupation2">Lawyer</label>
          </li>
          <li>
            <input id="occupation3" name="occupation3"element
                class="checkbox" type="checkbox" value="1" />
            <label for="occupation3">Teacher</label>
          </li>
          <li>
            <input id="occupation4" name="occupation4"
                class="checkbox" type="checkbox" value="1" />
            <label for="occupation4">Web designer</label>
          </li>
        </ol>
      </fieldset>
    </li>
  </ol>
</fieldset>
```

The **`label`** for the subgroup actually becomes the **`legend`** for the nested **`fieldset`**, then each of the checkboxes or radio buttons inside the **`fieldset`** receives its own **`label`**. The ordered list structure that was put in place at the top level is replicated on this sub-level as well, more for consistency than necessity although it can be very handy if you want to style some of the sub-items.

The nested elements will inherit the styles that we put in place for top-level items, so we'll have to set some rules specifically for nested elements before they'll display correctly:

element-subgroups.css (excerpt)

```
fieldset fieldset {
  margin-bottom: -2.5em;
  border-style: none;
  background-color: transparent;
  background-image: none;
}
fieldset fieldset legend {
  margin-left: 0;
  font-weight: normal;
}
fieldset fieldset ol {
  position: relative;
  top: -1.5em;
  margin: 0 0 0 11em;
  padding: 0;
}
fieldset fieldset label {
  float: none;
  width: auto;
  margin-right: auto;
}
```

Firstly, all the decoration on the nested **`fieldset`** is removed: **`background-color`**, **`background-image`**, and **`border`** properties. Instead, it's given a negative **`margin-bottom`** for the purposes of some trickery we'll see in a moment.

We want to make the **`legend`** look exactly like a normal **`label`**, so we remove the left margin and also take off its bold **`font-weight`**. It's important to be careful with the length of text inside the **`legend`**, as most browsers won't wrap the text in a **`legend`**. As a result, any **`width`** you've set for the legend/text will be ignored, as the text will just continue on in one line, possibly running over the rest of the **`form`**. We can overcome this limitation by exercising a maximum character width for the **`legend`** text and sizing the **`form`** columns in **`em`** units, so that with text-resizing the layout will scale accordingly.

NOTE ***Limitations of*** **`legend`**

Along with the inability of **`legend`** elements to wrap text, they are also resistant to **`width`** settings and text alignment. This use of **`legend`** elements for grouping within **`fieldset`** elements is only possible for left-aligned **`label`** elements, not right-aligned **`label`** elements.

We use the ordered list to position the nested form elements and **`label`** elements. Its left **`margin`** pushes the entire container away from the left edge, equivalent to the amount of **`margin`** given to form elements at the top level. Then, to bring the top of the form

elements in line with the top of their respective `legend`, we need to position the ordered list relatively and move it up by `-1.5em`. This will leave a large space at the bottom of the list (where the list would have been if it wasn't moved relatively), and this is where the `fieldset`'s negative `margin` comes into play. The negative `margin` pulls up the content after the `fieldset` by the same amount we moved the ordered list, making it look like there is no empty gap. The `padding` that's put on ordered lists at the top level isn't needed here, so we just set this property to `0`.

The last thing we need to do is to revert our `label` elements to their native state. This means we stop them from floating and set their `width` to `auto`. Because they're inline elements, they'll now sit nicely next to the actual form elements—checkboxes or radio buttons.

There's an additional change to make to the Internet Explorer-specific style sheet: to turn off the negative relative position on nested `legend`s. We don't have to deal with background colors on the nested `fieldset` elements, so the negative relative position isn't needed here:

element-subgroups-ie.css (excerpt)

```
fieldset fieldset legend {
  top: 0;
}
```

Once those new styles have been created, we end up with the form that appears in Figure 5.23—a nested `fieldset` that lines up perfectly with all the other form elements and gives the user a nice straightforward choice of options.

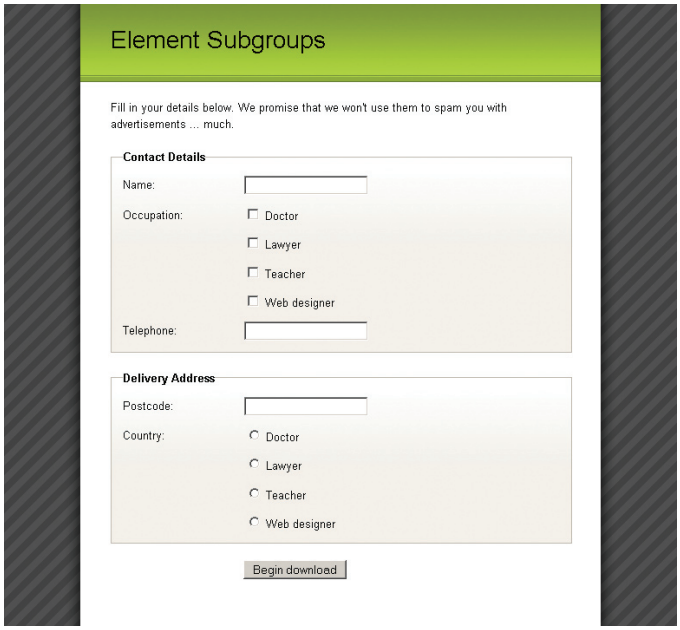


Figure 5.23: Nested subgroups of checkboxes and radio buttons

Required Fields and Error Messages

There are often little extra bits of information that you want to convey on a form, and they should be equally as accessible as the text `label` elements for the form element. In fact, to ensure that they're accessible, they should be included in the `label` itself. There are two types that we'll look at here: required fields and error messages.

Indicating Required Fields

The easiest and most accessible way of indicating the required fields on a form is to write "required" after the form `label`. This addition is not only read out by screenreaders, but it also means that an extra symbol key doesn't need to be provided for visual users, as is the case should you choose to mark required fields with an asterisk or other symbol.

To emphasize the importance of the information, we can add the text "required" inside an `em` element, which also gives us a stylable element to differentiate the "required" text from the `label` text:

required-fields.html (excerpt)

```
<label for="name">
  Name: <em>required</em>
</label>
```

To give the `em` its own little place on the `form`, we can set it to `display: block`, and change the appearance of the text:

required-fields.css (excerpt)

```
label em {
  display: block;
  color: #060;
  font-size: 85%;
  font-style: normal;
  text-transform: uppercase;
}
```

Our "required" markers now look like this:

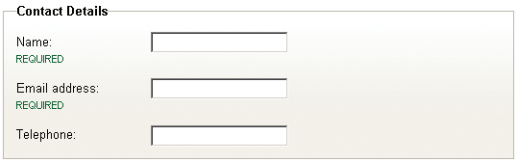


Figure 5.24: Form fields marked with textual "required" markers

However, the asterisk, or star, has now become a common tool for marking required fields, possibly due to its brevity. But it doesn't have much meaning outside the visual context—most screenreaders will read an asterisk character as "star." So you end up with a **`label`** being "Email address star"—a little confusing for the user.

For accessibility purposes, instead of including an actual asterisk character next to the form **`label`**, it's actually better to include an inline image of the asterisk, with **`alt`** text saying "required." This means that screenreader users will hear the word "required" instead of just "star," which is a lot more helpful. If you *are* using an image, you should include a key at the top of the **`form`** to let visual users know exactly what it means.

We still want to emphasize the fact that the **`label`** is required, so we just replace the text "required" inside the **`em`** element with the image of an asterisk:

required-fields-star1.html (excerpt)

```
<label for="name">
  Name: <em><img src="images/required_star.gif"
    alt="required" /></em>
</label>
```

This replacement doesn't actually need any styling; we can leave the **`em`** as an inline element and the asterisk will appear directly next to the form **`label`**:

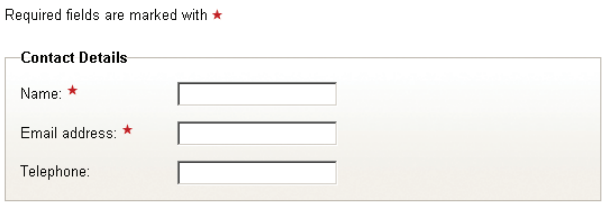


Figure 5.25: Inline asterisk marking required fields

Or, we can use some CSS to position the image absolutely and have it more closely associated with the form element itself:

required-fields-star2.css (excerpt)

```
label {
  position: relative;
  float: left;
  width: 10em;
  margin-right: 1em;
}
```

```
label em {
  position: absolute;
  left: 10em;
  top: 0;
}
```

When positioning the **em** absolutely, it's important to position its parent (the **label**) relatively, so that when we specify some coordinates for the **em**, they will be relative to the **label**'s top-left corner. The star image should be positioned in the gap between the **label** and the form element (created by the **label**'s right **margin**), so the value for the **em**'s **left** will depend upon what we've set there. Setting the top value for the **em** is just a precaution in case the image has wrapped onto a new line.

By taking this course of action, we'll end up with a much more orderly series of "required" markers, as shown in Figure 5.26.

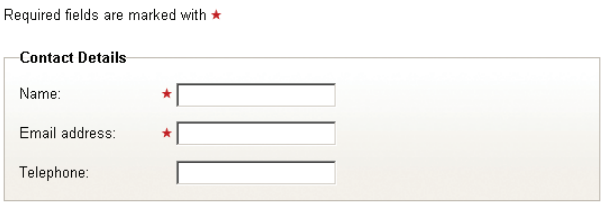


Figure 5.26: Required fields marked with absolutely positioned image of a star, aligned against form elements

Handling Error Messages

Error messages are handled in almost the same way as required markers. In order to be read out as a screenreader user places focus on the appropriate form element, they should form part of the **label**:

error-fields1.html (excerpt)

```
<label for="name">
  Email: <strong>This must be a valid email address</strong>
</label>
```

The semantic **strong** element is used to enclose the error message, distinguishing it from a required marker and giving it a stronger emphasis.

The styling is almost the same as it was for the textual "required" marker, except you might want to change the color. A good strong red is suitably alarming:

error-fields1.css (excerpt)

```
label strong {
  display: block;
  color: #C00;
  font-size: 85%;
  font-weight: normal;
  text-transform: uppercase;
}
```

This styling produces a layout such as that shown in Figure 5.27.

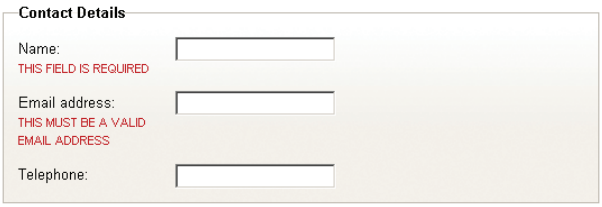


Figure 5.27: Error messages included as part of **`label`** element, displayed underneath the **`label`** text

An alternative placement of the error message does exist, but it depends upon a couple of prerequisites. The error message can be placed to the right of the form element as long as:

- The maximum width of any of the **`form`** elements is known.
- The error message is unlikely to wrap.

This placement involves the error message being positioned absolutely, so we must know in advance how far to move the error. Absolute elements are outside the flow of the document, so the other content will not adjust to accommodate the error message if it starts wrapping. If the design can be reconciled with these two problems, then the CSS for the job is:

error-fields2.css (excerpt)

```
label {
  position: relative;
  float: left;
  width: 10em;
  margin-right: 1em;
}
```

```
label strong {
  position: absolute;
  left: 27em;
  top: 0.2em;
  width: 19em;
  color: #C00;
  font-size: 85%;
  font-weight: normal;
  text-transform: uppercase;
}
```

Again, because the **strong** element is being positioned absolutely, its parent **label** must be positioned relatively to allow us to move the error message relative to the **label** itself.

The **width** of the error message is dictated by the space following the form element. The **left** is calculated by adding together the **width** of the form element, plus the **width** of the **label**, plus any extra space we need in order to align the error message properly.

Figure 5.28 shows how it ends up when viewed in the browser.

Figure 5.28: Error messages as part of the **label** element, displayed using absolute positioning

NOTE Inaccessible Error Text Solutions

It is possible to position the error text to the right of the text fields by changing the source order of the HTML. But this either:

- places the error text outside the **label**
- involves nesting the **form** element inside the **label** and placing the error text after the form element

Both of these solutions are inaccessible because screenreaders will most likely fail to read out the error message when the **form** element is focused.

In conjunction with right-positioning the error messages, we can also include error icons, to further highlight the problem areas on the form. The error icon is included in the HTML with an appropriate **alt** attribute:

error-fields3.html (excerpt)

```
<fieldset>
  <legend>Contact Details</legend>
  <ol>
    <li>
      <label for="name">
        Email: <strong><img src="images/error_cross.gif"
            alt="Error" /> This must be a valid email address
            </strong>
      </label>
      <input id="name" name="name" class="text" type="text" />
    </li>
```

We can now move it to the left of the form elements using absolute positioning. Because its parent (the **strong** element) is already absolutely positioned, any movement we make will be relative to that parent, so, effectively, we have to move it in a negative direction in order to shift it back over to the left:

error-fields3.css (excerpt)

```
label strong img {
  position: absolute;
  left: -16em;
}
```

This adjustment equates to the width of the form element, plus a little bit extra for spacing, so we'll get a nicely positioned icon, such as you can see in Figure 5.29.

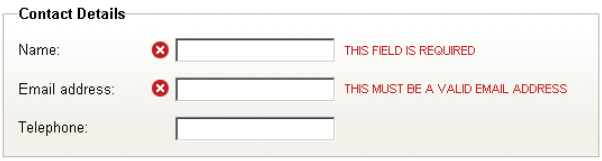


Figure 5.29: Error messages displaying to right of **form** elements, in combination with error icon on left

Summary

Now that you've finished this chapter, you have no excuse for producing inaccessible forms that use tables for positioning!

We've worked through the correct and effective labeling, grouping, layout, and styling of form elements, anticipating and solving potential problems of compatibility and

accessibility along the way. With the code provided here you've got quite a few different options for how you want your forms laid out, but there's still more you can do by combining and experimenting with different styles, form elements and layouts.

If there's an underlying message of this chapter, it's just to keep in mind that no matter what you do, your forms have to be usable and accessible above everything else. Forms, at the end of the day, are really all about your users being able to provide information and tell you what they want as easily as possible.

What's Next?

If you've enjoyed this chapter from *The Art & Science of CSS*, why not order yourself a copy?

This gorgeous, full-color book brings together a team of talented CSS authors who will show you how to use CSS to create designs that are not only standards-compliant, easy to maintain, and highly accessible, but are also visually stunning.

In the rest of the book, you'll:

- Learn to style images creatively: create galleries, thumbnails, and captions.
- Get creative with headings.
- Push the design envelope with innovative use of backgrounds.
- Build beautiful navigation: vertical, horizontal, and tabbed.
- Make your designs more fluid using fancy corner effects.
- Gain new-found respect for the table: make tabular data look amazing.
- And much more!

Each chapter was written by a renowned expert in the field and focuses on a particular building block of CSS-based design. Together, they show you how to bring your designs to life while retaining all the benefits of a fully standards-compliant web site.

The book's full-color layout and larger-than-normal size (8" x 10") help to show off the techniques demonstrated in the book.

This book is ideal for you if you want to gain the practical skills required to use CSS to make attractive web sites, especially if you're not the type who likes to learn by memorizing a formal specification and then trying to work out which browsers implemented it completely. The only knowledge you need to have is some familiarity with HTML. This book will give designers the skills they need to implement their ideas, and provides developers with creative inspiration through practical examples.

Order now and get it delivered to your doorstep!

Index

D

E

F

G

H

I

R

S

T

U

V

W

Y

Z