Extracted from:

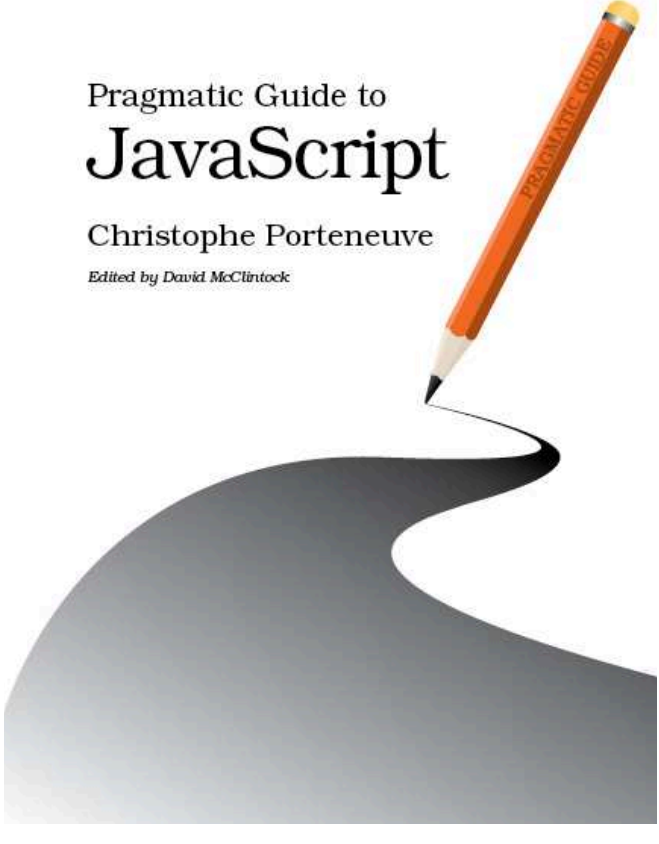
Pragmatic Guide to JavaScript

The Pragmatic Programmers

Pragmatic Guide to JavaScript

Christophe Porteneuve

Edited by David McClintock



18 Temporarily Disabling a Submit Button

Sometimes our forms take a while to be processed on the server side. Perhaps we're uploading a large file using a good ol' <input type="file"... /> field, or the server is just busy for some reason. At any rate, we do not want our user to resubmit the form while we're processing it already. Double-submits are just irritating, you know?

To avoid this, we can react to our form being submitted by *disabling* any UI means of submitting it, which boils down to <*input*> or <*button*> tags with type="submit" or type="image" attributes. Because a few browsers (such as our beloved MSIE) do not handle CSS attribute selectors well, we should "tag" those elements with a specific class, say *submit*, and use it to select elements we intend to disable.

On the facing page, the first script shows the minimum Prototype-based code for that. It's fairly straightforward.

You will likely want to go the extra mile and add some further UI decoration to your form as it is submitting; not all browsers render disabled fields in a clear visual style, and perhaps you also want to stress the fact that *something is going on*. (You're not just disabling that thing to be obnoxious, are you?)

The second script illustrates adding a custom class to our disabled <*input*> tags. Because the UI update that results from applying CSS is not as "built-in" as a disabling call, we also want to make sure our browser can "take a breath" before we have it submit the form (at which time it's likely to ignore any further visual update and just plow ahead with the submission). To solve this common problem, we delay( ) the submit( ) call by just a tenth of a second.

Also notice the that = this closure trick in our JavaScript code here. As you may know, calling a function (in this case, the one we end up delay( )ing) can lose our current *binding*—the value **this** refers to. Instead of forcing such a binding, which requires an extra layer of function wrapping and is therefore pretty costly, we rely on *closures* to let the code inside our ad hoc anonymous function retain a reference to our original **this** (the form being submitted) in order to call submit( ) on it in due time.[21]

21. Queasy about JS function bindings? Check out my ALA article at http://www.alistapart.com/articles/getoutbindingsituations/ for details. Not too sure about closures and how they're useful? My pal Juriy "Kangax" Zaytsev wrote a great article about it at http://msdn.microsoft.com/en-us/scriptjunkie/ff696765.aspx.

- Disable on the submit event.

Download form/submit/submit.js

```
function preventMultipleSubmits() {
  this.select('.submit').invoke('disable');
}

document.observe('dom:loaded', function() {
  $('commentForm').observe('submit', preventMultipleSubmits);
});
```

Download form/submit/index.html

```
<form id="commentForm" action="post_comment.php">
  <p>
    <label for="edtText">Your text</label>
    <textarea id="edtText" name="text" cols="40" rows="5"></textarea>
  </p>
  <p><input type="submit" class="submit" value="Send" /></p>
</form>
```

- Use classes for extra decoration (a bit of flourish).

Download form/submit/submit.js

```
function preventMultipleSubmits(e) {
  if (!this.hasClassName('submitting')) {
    e.stop();
  }
  this.addClassName('submitting').select('.submit').invoke('disable');
  var that = this;
  (function() { that.submit(); }).delay(0.1);
}
```

19 Providing Input Length Feedback

A common source of frustration when filling in forms is to suddenly see the text input stop dead, even when there was some text warning us of the maximum length. Not only that, but did you know that <*textarea*> has no maxlength= attribute? Seriously. It's not valid HTML and will be blissfully ignored (unless you're fortunate enough to be able to use HTML5).

So, to provide a unified way to specify maximum lengths, we can rely on dedicated CSS classes for, er, "data storage." They will use a two-part name: first a *maxLength* prefix, then a positive integer, stating the maximum length we want. See the markup on the facing page.

Then we can use JavaScript to do the following:

1. Dynamically decorate the form zones (I'll assume paragraphs, for the sake of brevity) containing these elements (the facing code adds a class to the paragraph), and then dynamically create the placeholder for remaining-length feedback.
2. Initialize the feedback zone to the current state.
3. Bind appropriate event listeners for as-you-type feedback.
4. Position the feedback zone (I put it under the bottom-right corner of its matching field here) and add it to the document, now that it's ready for prime time!

Now whenever typing occurs, we just need to update the feedback, and if we hit or exceed the maximum length (something impossible on a <*textarea*>), we'll backpedal to the maximum allowed length.

Note a couple of tricks in this code:

- We listen for both *keyup* and *keypress* in order to react to noncharacter keys (deletions, cuts, and pastes, mostly) and character keys. Listening to *keydown* would be useless because it occurs *before* the text changes, and we have no reliable way of determining across browsers and keyboard layouts *whether* the text will change.
- To avoid recomputing the maximum length at every keystroke, we cache it during setup. To associate maximum lengths with our fields, we use a JavaScript-based associative array between the fields' id= attributes[22] and the fields themselves. This is lighter weight than using expando properties.

22. We use Prototype's identify( ) here to make sure our element has an id=.

- Specify maximum lengths through markup.

Download form/feedback/index.html

```
<p>
  <label for="edtDescription">Description</label>
  <textarea id="edtDescription" name="description" cols="40"
   rows="5" class="maxLength200"></textarea>
</p>
```

- Set up feedback for maximum-length fields.

Download form/feedback/feedback.js

```
var maxLengths = {};

function bindMaxLengthFeedbacks() {
  var mlClass, maxLength, feedback;
  $$('*[class^=maxLength]').each(function(field) {
    field.up('p').addClassName('lengthFeedback');
    mlClass = field.className.match(/\bmaxLength(\d+)\b/)[0];
    maxLength = parseInt(mlClass.replace(/\D+/g, ''), 10);

    feedback = new Element('span', { 'class': 'feedback' });
    maxLengths[field.identify()] = [maxLength, feedback];
    updateFeedback(field);
    field.observe('keyup', updateFeedback).
      observe('keypress', updateFeedback);

    feedback.clonePosition(field, { setHeight: false,
      offsetTop: field.offsetHeight + 2 });
    field.insert({ after: feedback });
  });
}
```

- Give feedback on the fly.

Download form/feedback/feedback.js

```
function updateFeedback(e) {
  var field = e.tagName ? e : e.element();
  var current = field.getValue().length,
    data = maxLengths[field.id], max = data[0],
    delta = current < max ? max - current : 0;
  data[1].update('Remaining: ' + delta);
  if (current > max) {
    field.setValue(field.getValue().substring(0, max));
  }
}
```

The Pragmatic Bookshelf

The Pragmatic Bookshelf features books written by developers for developers. The titles continue the well-known Pragmatic Programmer style and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help you stay on top of your game.

Visit Us Online

Pragmatic Guide to JavaScript
http://pragprog.com/titles/pg_js
Source code from this book, errata, and other resources. Come give us feedback, too!

Register for Updates
http://pragprog.com/updates
Be notified when updates and new books become available.

Join the Community
http://pragprog.com/community
Read our weblogs, join our online discussions, participate in our mailing list, interact with our wiki, and benefit from the experience of other Pragmatic Programmers.

New and Noteworthy
http://pragprog.com/news
Check out the latest pragmatic developments, new titles and other offerings.

Buy the Book

If you liked this eBook, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: pragprog.com/titles/pg_js.

Contact Us

Online Orders:	www.pragprog.com/catalog
Customer Service:	support@pragprog.com
Non-English Versions:	translations@pragprog.com
Pragmatic Teaching:	academic@pragprog.com
Author Proposals:	proposals@pragprog.com
Contact us:	1-800-699-PROG (+1 919 847 3884)