CHAPTER 8

Get It Right, Bub: A JavaScript Validation Framework

When you hear the word *validation* in the context of a web application, you generally think of validating user input on a form. This usually evokes thoughts of writing event-handler code to perform various checks on form input before submitting it. All of this can quickly become rather messy. No matter how well you externalize your scripts and set up basic event-handler code that just calls functions, the simple fact is that these validations are scattered throughout your code—it's just a question of to what degree that's true.

Wouldn't it be great if we could truly externalize those validation checks to the point where we don't have to write any code at all? Wouldn't it be great if we could write a simple configuration file, say in XML, that defines what validations we want performed on each field and when? This is precisely the goal of the project in this chapter!

JSValidator Requirements and Goals

Building a JavaScript form validation framework isn't really that complex an undertaking, but making it truly useful requires a bit of forethought. To that end, let's lay out some of the goals we want to accomplish for our project, which we'll call JSValidator:

- The framework should be driven by an external configuration file written in XML. This will define what validations occur for which field, the parameters a given validation may need, and what to do if the validation fails.
- The framework should be extensible. We should be able to add validators—that is, classes that perform a given validation function—any time we want. We should be able to do this without modifying code, just the configuration file.
- We should create a couple of common validator types and cook them into the framework, so developers know they can always use these types without doing anything extra.
- This framework should be unobtrusive, meaning we shouldn't need to sprinkle code all over the place. Moreover, the form on which validations are defined should still work if JavaScript is turned off.

- Using the framework should be about as simple as can be for developers, requiring only a single JavaScript import, some minimal configuration parameters, and the external configuration file.
- The framework should offer error reporting in three ways: `alert()` messages, messages inserted into a specified `<div>`, and highlighting of fields (with developer-defined styles). Moreover, the messages should allow for reuse by understanding a token system, where the tokens can be replaced with values defined in the configuration file when a validation failure occurs.

If that sounds like a lot of work, I think you'll be surprised to see that it isn't quite as much as you might think. You'll also find that the end result is something useful and expandable that can be applied in your projects immediately. But I've left the door open for a couple of enhancements that will make it even more useful, and applying them will give you some valuable exercises to work through in order to expand your JavaScript chops.

With our goals and requirements set, let's see how we'll make it work!

How We Will Pull It Off

As mentioned in the requirements, we want JSValidator to use an external configuration file to define all the validations that will be performed for a given HTML form. More specifically, we want this configuration file to be in the form of XML.

There are always choices to be made when deciding on a format for a configuration file, but I feel that while XML isn't perfect for everything, for configuration files, it probably is. That is because it tends to be self-describing (unless the developer does a bad job laying it out). Also, size and parse time don't generally matter, because you tend to parse them once at startup and not frequently after that. You can take a slight performance hit at startup usually, which also means you don't care so much how large or verbose the file is. In fact, the more verbose, the better, most likely, as long as that verbosity adds to the clearness of the configuration.

So, having decided XML is the way to go for JSValidator, on to the next question: how are we going to deal with parsing it? If you've ever done XML parsing in JavaScript, you know that it isn't necessarily a pleasant experience. I discussed this a bit in Chapter 7, so no need to rehash the pain. In Chapter 7, we also find the answer to making it a bearable situation: JSDigester. With JSValidator, you'll see a real-world usage of JSDigester. You'll see how it allows you to take a somewhat complex XML document, describe it with a few simple rules, and have it parsed into JavaScript object quickly and easily. (If you skipped the details on JSDigester in Chapter 7, I suggest going back and reading that now.)

Now that you know how we're going to parse the configuration file, let's answer another important question: how are we going to load it? Remember one of the other goals for JSValidator is to make it as simple and unobtrusive for a developer to use as possible. As a matter of fact, here's all the developer has to do on any given page to use the framework:

```
<script>
  var JSVConfig = {
    pathPrefix : "jsvalidator/",
    configFile : "jsv_config.xml",
    manualInit : false
  };
```