Validating the date format

There are plans to adapt the jQuery UI Datepicker widget for use in a jQuery Mobile site. At the time of this writing, the widget was still highly experimental. When a stable version becomes available, you might prefer to use it instead of the following solution, which checks only the date format, not the validity of the date.

This solution uses a complex *regular expression.* Copying regular expressions from printed books often leads to mistakes, so I suggest that you copy the code for the regex from reservations_date.html in ch06/complete.

1. Insert the following code inside the jQuery document-ready function that you created in the previous section (the change-handler script for the Select Menu widget has been omitted for brevity):

```
$(function() {
    // Change handler for Select Menu widget
    . . .
    // Check date format
    $('#arrival, #departure').live('blur',
    ➥ function(e) {

    });
});
```

This creates a jQuery selector for the arrival and departure text input fields, and binds an onblur event handler to both of them.

2. Inside the new event-handler function, define a variable to hold the regex. Because of the limitations of the printed page, it's split over two lines, but it must be on a single, unbroken line in your code:

```
var pattern = /^(1[0-2]|0[1-9])\/(3[01]|
➥ [12][0-9]|0[1-9])\/201[1-9]$/;
```

This matches a date in the MM/DD/YYYY format in the years 2011–2019. The date must include leading zeros.

CLOSE-UP

Regular Expressions

A regular expression—or regex for short—is a pattern for matching text. Regexes use a combination of literal text and characters with special meanings. For example, \w matches any alphanumeric character or an underscore. Learning how to build regexes isn't easy, but it's a skill worth acquiring if you use a lot of JavaScript or other programming languages. You'll also find regexes extremely useful in the Dreamweaver Find and Replace dialog box. For a basic introduction to regexes, see my tutorial at www.adobe.com/devnet/dreamweaver/articles/regular_expressions_pt1.html.

To match a date in the European DD/MM/YYYY format, use the following pattern instead:

```
var pattern = /^(3[01]|[12][0-9]|0[1-9])\/
➥ (1[0-2]|0[1-9])\/201[1-9]$/;
```

To match a date in the YYYY/MM/DD format commonly used in East Asia, use this:

```
var pattern = /^201[1-9]\/(1[0-2]|0[1-9])\/
➥ (3[01]|[12][0-9]|0[1-9])$/;
```

3. You need to obtain the value entered by the user using `$(this).val()` and pass it as an argument to the JavaScript `test()` method, which returns `true` or `false` depending on whether the value matches the regex. If the input doesn't match, display an alert. The completed code looks like this:

```
$(function() {
    // Change handler for Select Menu widget
    . . .
    // Check date format
    $('#arrival, #departure').live('blur',
    function(e) {
        var pattern = /^(1[0-2]|0[1-9])\/(3
        ➥ [01]|[12][0-9]|0[1-9])\/201[1-9]$/;
        if (!pattern.test($(this).val())) {
            alert('Date must be in MM/DD/YYYY
            ➥ format with leading zeros');
        }
    });
});
```

The exclamation mark after the opening parenthesis of the condition is the logical NOT operator, which negates the meaning of the following expression. In other words, the condition means "if the value doesn't match the pattern."

Checking that all required fields have been filled in

Because you can't be sure that all browsers recognize the HTML5 `required` attribute, it's a good idea to check that required fields aren't blank before submitting the form.

If any fields are missing, you can dynamically add a warning message to the form, highlight the affected fields, and prevent the form from being submitted.

1. Add the following style rule to the <style> block in the <head> of reservations.html:

```
.invalid {
    background-color: rgba(255,0,0,0.1);
}
```

This uses the *rgba()* color format to set the background color to red with 10% alpha transparency—in effect, pale pink.

2. Add the following code inside the jQuery document-ready function at the bottom of the page:

```
$(function() {
    // Change handler for Select Menu widget
    . . .
    // Check date format
    . . .
    // Check required fields
    $('form').live('submit', function(e) {
        // Initialize Boolean variable
        var OK = true;
        // Test each required field
        $('input[required]').each(function() {
            if ($.trim($(this).val()) == '') {
                // If blank, set OK to false
                OK = false;
                // Style current element
                $(this).addClass('invalid');
            }
        });
        if (!OK) {
            // Add warning if field(s) blank
            $('form').before('<p id="errors">
            ➥ Please fill in required fields.
            ➥ </p>');
            // Scroll to top of page
            window.scrollTo(0,0);
```

```
            } else {
                // Remove warning if OK
                $('#errors').remove();
            }
            return OK;
        });
    });
```

This binds an onsubmit event handler to the form. It must be bound to the form instead of the submit button because of the way that jQuery Mobile handles form submission. The script begins by initializing a Boolean variable OK, which the event handler ultimately returns. If any required fields are blank, OK is reset to false, which prevents the form from being submitted.

The each() method is applied to the attribute selector $('input[required]'). This executes the same function on each required field. The condition inside the function uses the jQuery $.trim() utility method to strip leading and trailing spaces from the user input, and compares the value with an empty string. If the field is blank, OK is reset to false, and the invalid class is applied to the current element.

If OK is false, the before() method prepends a warning paragraph to the form, and the page is scrolled to the top to bring the warning into view. The paragraph is given an ID so that it can be removed later if the form is submitted without errors.

3. There's just one final refinement: Add the following code inside the jQuery document-ready handler:

```
$(function() {
    // Change handler for Select Menu widget
    . . .
    // Check date format
    . . .
    // Check required fields
    . . .
    // Clear invalid class on focus
    $('input[required]').live('focus',
    ↪ function(e) {
```

```
        $(this).removeClass('invalid');
    });
});
```

This binds an onfocus event handler to the required fields to remove the invalid class when the focus moves into the field. Of course, this is needed only on required fields that were left blank when the form was submitted, but the function fails silently if the class hasn't been applied to an element.

4. Before you can test the page in Live view, you need to add data-ajax to the opening <form> tag and set its value to false.

5. Activate Live view and scroll down to the Submit button without filling in any fields. Hold down the Ctrl/Command key and click Submit. The page should scroll to the top and display the warning message. The blank fields should also be highlighted in pale red (**Figure 6.1**).

Figure 6.1 The form now warns users about required fields that have been left blank.

Live view does not clear the invalid class from required fields when you give them focus. However, the fields are cleared as expected in a browser.