

JavaScript: Form Validation: Date and Time

4 Tweets <https://twitter.com/intent/tweet?text=Form+Validation%3A+Date+and+Time&url=http%3A%2F%2Fwww.the-art-of-web.com%2Fjavascript%2Fvalidate-date%2F&via=theartofweb>

14 Likes <http://www.facebook.com/sharer/sharer.php?u=http%3A%2F%2Fwww.the-art-of-web.com%2Fjavascript%2Fvalidate-date%2F>

0 Shares <https://plus.google.com/share?url=http%3A%2F%2Fwww.the-art-of-web.com%2Fjavascript%2Fvalidate-date%2F>

Feedback

When capturing information for insertion into a database, or use in other processing, it's important to control what the user can enter. Otherwise you can end up with values in the database that have no relation to reality.

1. Checking for valid format

In this example, the date fields will only accept input that matches the pattern 'dd/mm/yyyy' (this could just as easily be changed to 'yyyy-mm-dd' or 'mm/dd/yyyy'). The time field will allow input starting with 'hh:mm' following by an optional 'am' or 'pm'. The fields can also be empty.

Event Details

Start Date:
 (dd/mm/yyyy)

Start Time:
 (eg. 00:51 or 12:51am)

Submit

The code behind the form is as follows:

```
<script type="text/javascript">

function checkForm(form)
{
    // regular expression to match required date format
    re = /^d{1,2}\d{1,2}\d{4}$/;

    if(form.startdate.value != '' && !form.startdate.value.match(re)) {
        alert("Invalid date format: " + form.startdate.value);
        form.startdate.focus();
        return false;
    }

    // regular expression to match required time format
    re = /^d{1,2}:\d{2}([ap]m)?$/;

    if(form.starttime.value != '' && !form.starttime.value.match(re)) {
        alert("Invalid time format: " + form.starttime.value);
        form.starttime.focus();
        return false;
    }

    alert("All input fields have been validated!");
    return true;
}

</script>
```

For each field in the form (first the dates, then the time field), a check is made as to whether the input is blank. If not, the input is compared to the regular expression. The expressions use a pre-defined class `\d` which represents any numeric character (0-9).

If you wanted to be really finicky the regular expression to match a date could also be written as:

```
re = /^[0-3]?[0-9]\[/[01]?[0-9]\[/[12][90][0-9][0-9]$/
```

If the input doesn't match the regular expression then an error message is presented, the routine stops the form from submitting by returning a false value and the focus is moved to the relevant form field.

If all tests are passed, then a value of true is returned which enables the form to be submitted.

This routine DOES NOT check that the date or time input values are valid, just that they match the required format (d/m/y and h:m). Read further for more comprehensive checking.

2. Checking that input values are within bounds

Once you're in control of the input format, it's a lot easier to check that the values are actually valid. The function has been improved now so that the day, month and year values are checked to ensure that they're in the right ball-bark (ie. 1-31 for the day and 1-12 for the month). Also the year must be between 1902 and the current year.

The year limitation would be used if you were asking for a date of birth or date of some recent event. If you're setting up a calendar of future events you would check that the year is the current year or greater.

Event Details

Start Date:
 (dd/mm/yyyy)

Start Time:
 (eg. 00:51 or 12:51am)

Submit

The code behind the form now is as follows:

```
<script type="text/javascript">

function checkForm(form)
{
    // regular expression to match required date format
    re = /^(d{1,2})\/(d{1,2})\/(d{4})$/;

    if(form.startdate.value != '') {
        if(regs = form.startdate.value.match(re)) {
            // day value between 1 and 31
            if(regs[1] < 1 || regs[1] > 31) {
                alert("Invalid value for day: " + regs[1]);
                form.startdate.focus();
                return false;
            }
            // month value between 1 and 12
            if(regs[2] < 1 || regs[2] > 12) {
                alert("Invalid value for month: " + regs[2]);
                form.startdate.focus();
                return false;
            }
            // year value between 1902 and 2015
            if(regs[3] < 1902 || regs[3] > (new Date()).getFullYear() [http://developer.mozilla.org/en/docs/Core_JavaScrip
                alert("Invalid value for year: " + regs[3] + " - must be between 1902 and " + (new Date()).getF
                form.startdate.focus();
                return false;
            }
        }
    }
}
```

[expand code box](#)

If you're not already familiar with regular expressions, then this might be getting a bit complicated. Basically, for each of the regular expression tests, an array is returned holding each component of the pattern that we've matched.

For example, when the date is checked, the return value, `regs`, is an array with elements 1 through 3 containing the day, month and year components of the input string. For the time check, the array returned includes the hour (pos 1), minutes (pos 2) and, optionally, the am/pm string (pos 3).

Each of these values is then tested against an allowed range (days: 1 - 31; months: 1 - 12; years: 1902 - 2015; and so on).

This script only confirms that the input format is correct and that each individual value falls within its allowed range. It does not check for leap years or invalid dates at the end of short months.

3. Modular checkDate() function

As we've seen before, creating re-usable functions can significantly reduce the size of your JavaScript code. These functions can even be included from an external javascript file so that the browser can cache them, and so the programmer isn't always copying and pasting.

In this case, we've created a stand-alone functions which will validate a date field:

```
// Original JavaScript code by Chirp Internet: www.chirp.com.au
// Please acknowledge use of this code by including this header.

function checkDate(field)
{
    var allowBlank = true;
    var minYear = 1902;
    var maxYear = (new Date()).getFullYear();

    var errorMsg = "";

    // regular expression to match required date format
    re = /^(d{1,2})\/(d{1,2})\/(d{4})$/;

    if(field.value != '') {
        if(regs = field.value.match(re)) {
            if(regs[1] < 1 || regs[1] > 31) {
                errorMsg = "Invalid value for day: " + regs[1];
            } else if(regs[2] < 1 || regs[2] > 12) {
                errorMsg = "Invalid value for month: " + regs[2];
            } else if(regs[3] < minYear || regs[3] > maxYear) {
                errorMsg = "Invalid value for year: " + regs[3] + " - must be between " + minYear + " and " + maxYear;
            }
        } else {
            errorMsg = "Invalid date format: " + field.value;
        }
    } else if(!allowBlank) {
        errorMsg = "Empty date not allowed!";
    }
}
```

```
if(errorMessage != "") {
    alert(errorMessage);
    field.focus();
    return false;
}

return true;
}
```

Dealing with alternative date formats is simple enough:

- To convert to mm/dd/yyyy (US format) just swap regs[1] and regs[2] in the above code.
- To convert to yyyy-mm-dd (European format) you need to change the regexp to `/^\d{4})-(\d{1,2})-(\d{1,2})$/` and swap regs[1] and regs[3].

This is explained again below.

4. Modular checkTime() function

And a function that will validate a time input:

```
// Original JavaScript code by Chirp Internet: www.chirp.com.au
// Please acknowledge use of this code by including this header.

function checkTime(field)
{
    var errorMessage = "";

    // regular expression to match required time format
    re = /^(\d{1,2}):(\d{2})(:00)?([ap]m)?$/;

    if(field.value != '') {
        if(regs = field.value.match(re)) {
            if(regs[4]) {
                // 12-hour time format with am/pm
                if(regs[1] < 1 || regs[1] > 12) {
                    errorMessage = "Invalid value for hours: " + regs[1];
                }
            } else {
                // 24-hour time format
                if(regs[1] > 23) {
                    errorMessage = "Invalid value for hours: " + regs[1];
                }
            }
            if(!errorMessage && regs[2] > 59) {
                errorMessage = "Invalid value for minutes: " + regs[2];
            }
        } else {
            errorMessage = "Invalid time format: " + field.value;
        }
    }

    if(errorMessage != "") {
        alert(errorMessage);
        field.focus();
        return false;
    }

    return true;
}
```

5. Using these functions for validation

It's now much easier to see what the main validation function is doing:

```
<script type="text/javascript">

    function checkForm(form)
    {
        if(!checkDate(form.startdate)) return false;
        if(!checkTime(form.starttime)) return false;
        return true;
    }

</script>
```

In each case the value passed to the function is the form field rather than the field value. The output will be almost identical to the earlier examples.

In this simple example we can even rewrite the checkForm function above as:

```
<script type="text/javascript">

    function checkForm(form)
    {
        return checkDate(form.startdate) && checkTime(form.starttime);
    }

</script>
```

The associated HTML form would be as follows:

```
<form method="POST" action="..." onsubmit="return checkForm(this);">
<fieldset>
<legend>Event Details</legend>
<label>Start Date: <input type="text" size="12" placeholder="dd/mm/yyyy" name="startdate"> <small>(dd/mm/yyyy)
<label>Start Time: <input type="text" size="12" name="starttime"> <small>(eg. 18:17 or 6:17pm)</small></label>
<p><input type="submit"></p>
</fieldset>
</form>
```

6. Adding HTML5 input validation

There's now no excuse for having forms without [HTML5 form validation](#) attributes. Notably the pattern and required attributes which allow the browser to perform it's own validation:

```
<form method="POST" action="..." onsubmit="return checkForm(this);">
<fieldset>
<legend>Event Details</legend>
<label>Start Date: <input type="text" size="12" required pattern="\d{1,2}/\d{1,2}/\d{4}" placeholder="dd/mm/"
<label>Start Time: <input type="text" size="12" pattern="\d{1,2}:\d{2}([ap]m)?" name="starttime"></label>
<p><input type="submit"></p>
</fieldset>
</form>
```

In practice most modern browsers will now use HTML form validation to preempt any JavaScript validation - with the notable exception of Safari.

Event Details

Start Date:
 (dd/mm/yyyy)

Start Time:
 (eg. 18:17 or 6:17pm)

Submit

Read more about [HTML5 form validation](#).

7. Adjusting the code for different date formats

Visitors from some countries may find it confusing that we're using the dd/mm/yyyy date format instead of the American or other standards. Modifying the code involves only minor changes:

US Date Format MM/DD/YYYY

In the checkDate function above you only need to replace references to regs[1] with regs[2] and vice-versa to reflect the change in order of the day and month values.

European Format YYYY-MM-DD

In the checkDate function above you only need to change the regular expression (re) to `/^\d{4})-(\d{1,2})-(\d{1,2})/` and then replace references to regs[1] with regs[3] and vice-versa as the year and day values have now changed position.

8. Checking month endings and leap years

In JavaScript to check for different month lengths, particularly for February in leap years, you need quite a bit of extra code. I'm not going to show that here, but you can find a link to get started under References below.

Instead we're going to make use of [Form Validation using Ajax](#) to do some real-time checking using a server-side PHP script to get a definitive answer.

Ajax Date Validation

Date of Birth:

Check Date Validity

When you enter a date in the format dd/mm/yyyy the value is sent via an Ajax call to the server where it is validated using the PHP [checkdate](#) (<http://www.php.net/checkdate>) function.

The return value is displayed next to the input field:

Ajax Date Validation	Ajax Date Validation
Date of Birth: <input type="text" value="22/4/2013"/> date 22/4/2013 is valid <input type="button" value="Check Date Validity"/>	Date of Birth: <input type="text" value="31/2/2013"/> date 31/2/2013 is not valid <input type="button" value="Check Date Validity"/>

Other actions could also be taken such as disabling form submission until there is a valid date.

The relevant portions of the HTML are as follows:

```
<script type="text/javascript" src="ajaxrequest.js [http://www.chirp.com.au/ajaxrequest.js]"></script>
<script type="text/javascript">

function callAjax(value, target)
{
    if(encodeURIComponent) {
        var req = new AjaxRequest();
        var params = "value=" + encodeURIComponent(value) + "&target=" + target;
        req.setMethod("POST");
        req.loadXMLDoc("valid_date.php", params);
    }
}

</script>

...

<form ...>

...

<p>Date of Birth: <input id="field_dob" name="dob" onchange="
    if(this.value.match(/^\d{1,2})\/(\d{1,2})\/(\d{4})$/) {
        callAjax(this.value, this.id);
    }
"> <small id="rsp_field_dob"><!-- --></small></p>

...

</form>
```

The JavaScript onchange event handler simply passes the date input value to a server-side script valid_date.php where it will be tested. It then waits for an XML response containing instructions as to what message to display.

The PHP script valid_date.php simply breaks up the date input string and passes the values to the built-in checkdate() function - which presumably knows all about leap years and other strange features of the Gregorian calendar:

```
<?PHP
include "class.xmlresponse.php";

if($_POST) {
    $target = trim($_POST['target']);
    $value = trim($_POST['value']);
} else {
    die("Error: missing POST values");
}

$date_is_valid = false;
if(preg_match("@(\d{1,2})\/(\d{1,2})\/(\d{4})@", $value, $regs)) {
    list($tmp, $day, $month, $year) = $regs;
    $date_is_valid = checkdate($month, $day, $year);
}

$retval = $date_is_valid ? "date $value is valid" : "date $value is not valid";

$xml = new xmlResponse();
$xml->start();

$xml->command('setContent',
    array('target' => "rsp_$target", 'content' => htmlentities($retval))
);
if($date_is_valid) {
    $xml->command('setstyle',
        array('target' => "rsp_$target", 'property' => 'color', 'value' => 'green')
    );
} else {
    $xml->command('setstyle',
        array('target' => "rsp_$target", 'property' => 'color', 'value' => 'red')
    );
    $xml->command('focus', array('target' => $target));
}
$xml->command('setstyle',
    array('target' => "form1", 'property' => 'cursor', 'value' => 'default')
);

$xml->end();
?>
```

You can copy the code for valid_date.php from the box below:

```
<?PHP
include "class.xmlresponse.php";

if($_POST) {
    $target = trim($_POST['target']);
```

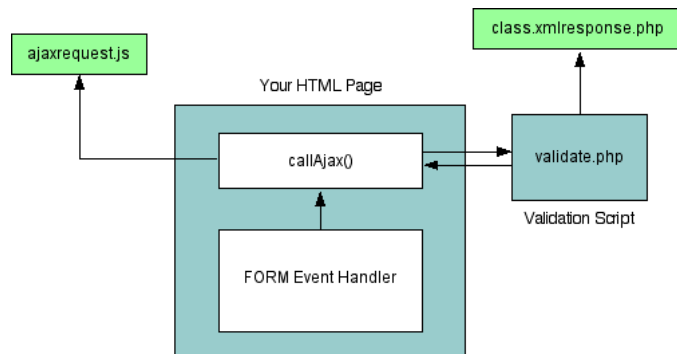
The script class.xmlresponse.php was introduced in [Generating an XML Response for Ajax Applications](#) and can also be copied below:

```
<?PHP
// Original PHP code by Chirp Internet: www.chirp.com.au
// Please acknowledge use of this code by including this header.

class xmlResponse
```

Both files need to be saved in the same folder as your HTML page for the date validation to work - so you have four files in total - the original HTML file with the form, ajaxrequest.js, class.xmlresponse.php and valid_date.php.

Similar to this setup from another example:



With Ajax you can make use of more powerful server-side functions and don't have to include large JavaScript libraries for validating dates and other values. Just be aware that it means more requests to the server which can be slower than downloading and running JavaScript code.

The AjaxRequest class is a simple one we've created and use on a number of projects. You can find the details in [Web Services using XMLHttpRequest \(Ajax\)](#) and related articles.

9. References

- [Regular Expression \(and String\) Methods](http://www.webreference.com/js/column5/methods.html) [http://www.webreference.com/js/column5/methods.html]
- [Is it Leap Year?Using Javascript](http://www.codeproject.com/KB/scripting/Leap_Year_JavaScript.aspx) [http://www.codeproject.com/KB/scripting/Leap_Year_JavaScript.aspx]

10. Related Articles - Form Validation

- [HTML5 Form Validation Examples](#) [HTML]
- [Validating a checkbox with HTML5](#) [HTML]
- [Password Validation using regular expressions and HTML5](#) [JAVASCRIPT]
- [Date and Time](#) [JAVASCRIPT]
- [Preventing Double Form Submission](#) [JAVASCRIPT]
- [Form Validation](#) [JAVASCRIPT]
- [Credit Card numbers](#) [JAVASCRIPT]
- [A simple modal feedback form with no plugins](#) [JAVASCRIPT]
- [Protecting forms using a CAPTCHA](#) [PHP]
- [Basic Form Handling in PHP](#) [PHP]

11. User Comments

Max 20 October, 2012

I found a small bug in your checkTime() function.
It doesn't allow a space after minutes and before the am/pm
example: "03:30 am" will get an invalid time.
while "03:30am" will be ok.

I modified the reg to the following:
re = /^(\d{1,2}): (\d{2})(:00)??([ap]m)?\$/;

Thanks,

pradeep 17 September, 2011

Thanks for the code... It helped me a lot.... I had many date fields in my form.... This helped me a lot.... I just modified the code to my need... that was a small thing.... Thanks for the code again.....

Ted Schumacher 28 June, 2007

There is no explanation of the string ([ap]m). I have done google searches but cannot find an explanation. Can you help?

It's a regular expression "([ap]m)?" that captures an (optional) 'am' or 'pm' at the end of the input:

- [ap] matches a single 'a' or 'p'
- [ap]m then matches 'am' or 'pm'
- ([ap]m) also 'captures' the value
- ([ap]m)? makes the match and capture 'optional'

your date script is allowing invalid dates like 29/2/2007 29 feb doesnt exist. kindly do validate the end dates of months. and leap years.

It was made clear in the article that that level of checking is not included. I suggest using a server-side script such as the [PHP checkdate function](http://www.php.net/checkdate) (<http://www.php.net/checkdate>) which can be called using Ajax or other means.