Constraint Validation

Native Client Side Validation for Web Forms

Introduction

- Before HTML5 there was no means of implementing client side validation natively.
- The developers have appealed to a variety of JavaScript based solutions.
- HTML5 introduced a new concept known as constraint validation.
- It's a native means of implementing client side validation on web forms.

Constraint Validation

- The constraint validation is an algorithm that browsers run when a form is submitted to determine its validity.
- This algorithm utilizes new HTML5 attributes such as min, max, step, pattern, and required.
- It uses as well as existing attributes such as maxlength and type.

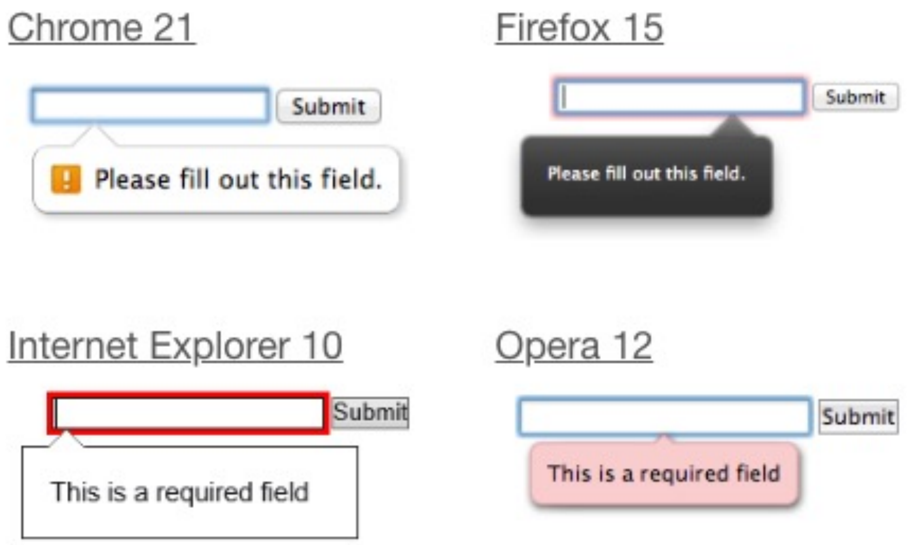
Constraint Validation

In this form we've included a required text input.

```
<form>
    <input type="text" required value="" />
    <input type="submit" value="Submit" />
</form>
```

Constraint Validation

If we attempt to submit this form, browsers will prevent the submission and display a message.



Chrome 21

Please fill out this field.

Firefox 15

Please fill out this field.

Internet Explorer 10

This is a required field

Opera 12

This is a required field

Constraint Validation

- According to HTML5, how errors are presented to the user is left up to the browser itself.
- HTML5 spec does provide a full DOM API, new HTML attributes, and CSS hooks we can use to customize the user experience.

DOM API

The constraint validation API adds some properties and methods to DOM nodes.

- The willValidate property indicates whether the node is a candidate for validation.
- For submittable elements this will be set to true unless for some reason the node is restricted from validation.

DOM API

Because an input element has the disabled attribute it will be restricted from validation.

```
<input type="text" id="foo" />
<input type="text" id="bar" disabled />

<script>
  document.getElementById('foo').willValidate; // true
  document.getElementById('bar').willValidate; // false
</script>
```

DOM API

- The validity property of a DOM node returns a ValidityState object
- It contains a number of boolean properties related to the validity of the data in the node.

DOM API

The patternMismatch property is true if the node's value does not match its pattern attribute.

```
<input id="foo" pattern="[0-9]{4}" value="1234" />
<input id="bar" pattern="[0-9]{4}" value="ABCD" />

<script>
  document.getElementById('foo')
        .validity.patternMismatch; // false
  document.getElementById('bar')
        .validity.patternMismatch; // true
</script>
```

DOM API

The rangeOverflow property is true if the node's value is greater than its max attribute.

```
<input id="foo" type="number" max="2" value="1" />
<input id="bar" type="number" max="2" value="3" />

<script>
   document.getElementById('foo')
         .validity.rangeOverflow; // false
   document.getElementById('bar')
         .validity.rangeOverflow; // true
</script>
```

DOM API

The stepMismatch property is true if the node's value is invalid per its step attribute.

```
<input id="foo" type="number" step="2" value="4" />
<input id="bar" type="number" step="2" value="3" />

<script>
  document.getElementById('foo')
        .validity.stepMismatch; // false
  document.getElementById('bar')
        .validity.stepMismatch; // true
</script>
```

DOM API

The typeMismatch property is true if an input node's value is invalid per its type attribute.

```
<input id="foo1" type="url" value="http://foo.com" />
<input id="bar1" type="url" value="http://foo" />

<input id="foo2" type="email" value="foo@foo.com" />
<input id="bar2" type="email" value="foo.com" />
```

DOM API

```
<script>
    document.getElementById('foo1') // http://foo.com
            .validity.typeMismatch; // false
    document.getElementById('bar1') // http://foo
            .validity.typeMismatch; // true
    document.getElementById('foo2') // foo@foo.com
            .validity.typeMismatch; // false
    document.getElementById('bar2') // foo.com
            .validity.typeMismatch; // true
</script>
```

DOM API

The valueMissing property is true if the node has a required attribute but has no value.

```
<input id="foo" type="text" required value="foo" />
<input id="bar" type="text" required value="" />

<script>
  document.getElementById('foo')
        .validity.valueMissing; // false
  document.getElementById('bar')
        .validity.valueMissing; // true
</script>
```

DOM API

The valid property is true if all of the validity conditions are false.

```
<input id="foo1" type="text" required value="foo" />
<input id=" foo2" type="text" required value="" />

<input id="bar1" type="number"
        required step="2" value="4" />
<input id="bar2" type="number"
        required step="2" value="3" />
```

DOM API

```
<script>
   document.getElementById('foo1') // value="foo"
         .validity.valid; // true
   document.getElementById('foo2') // value=""
         .validity.valid; // false
   document.getElementById('bar1') // step="2" value="4"
         .validity.valid; // true
   document.getElementById('bar2') // step="2" value="3"
         .validity.valid; // false
</script>
```

DOM API

When we use the checkValidity() method:

- On a form element node (input, select, textarea), it returns true if the element contains valid data.
- On a form node, it returns true if all of the form's children contain valid data.

DOM API

```
<form id="form1">
    <input id="input1" type="text" />
</form>

<form id="form2">
    <input id="input2" type="text" />
    <input id="input3" type="text" required />
</form>
```

DOM API

```
<script>
   document.getElementById('form1')
         .checkValidity();  // true
   document.getElementById('input1')
         .checkValidity(); // true
   document.getElementById('form2') // required
         .checkValidity();  // false
   document.getElementById('input2')
         .checkValidity(); // true
</script>
```

DOM API

- Every time a form element's validity is checked via checkValidity() and fails, an invalid event is fired for that node.
- We could then run some code whenever the node was checked and contained invalid data.

```
document.getElementById('input1')
        .addEventListener('invalid', function() {
    // field contains invalid data
}, false);
```

DOM API

We can also use the change event for notifications of when a field's validity changes (there is no valid event).

```
document.getElementById('input1')
        .addEventListener('change', function(event) {

    if (event.target.validity.valid) {
        // field contains valid data
    } else {
        // field contains invalid data
    }
}, false);
```

DOM API

- The validationMessage property contains the message the browser displays to the user when a node's validity is checked and fails.
- If the DOM node is not a candidate for constraint validation or if it contains valid data, then validationMessage will be set to an empty string.

DOM API

- The setCustomValidity() method changes the validationMessage property as well as allows us to add custom validation rules.
- Setting this property passing in an empty string, marks the field as valid; and passing any other string, marks the field as invalid.
- The customError property will true if a custom validity message has been set per a call to the setCustomValidity() method.

DOM API

```
<input id="foo" />

<script>
   document.getElementById('foo')
          .validity.customError; // false
   document.getElementById(foo')
          .setCustomValidity('Invalid field !');
   document.getElementById(foo')
          .validity.customError; // true
</script>
```

DOM API

In this example, we had two password fields and we wanted to enforce be equal.

```
if (document.getElementById('password1').value !=
    document.getElementById('password2').value) {

    document.getElementById('password1') // invalid
            .setCustomValidity('Must match !'); // error msg
} else {
    document.getElementById('password1') // valid
            .setCustomValidity(''); // no error msg
}
```

HTML Attributes

- The boolean novalidate attribute can be applied to form nodes.
- When present it indicates that the form's data should not be validated when it is submitted.
- The boolean formnovalidate attribute can be applied to button and input nodes to prevent form validation.

HTML Attributes

Because this form has this attribute it will submit even though it contains an empty required input.

```
<form novalidate>
    <input type="text" required />
    <input type="submit" value="Submit" />
</form>
```

HTML Attributes

- When "Validate" button is clicked form, submission will be prevented because of the empty input.
- When "Send" button is clicked, form will submit despite invalid data because of the formnovalidate attribute.

```
<form>
   <input type="text" required />
   <input type="submit" value="Validate"/>
   <input type="submit" formnovalidate value="Send"/>
</form>
```

CSS Hooks

- Writing effective form validation is not just about the errors themselves.
- It's also about to show the errors to the user in a usable way.
- :valid pseudo-class will match form elements that meet their specified constraints and :invalid will match those that do not.

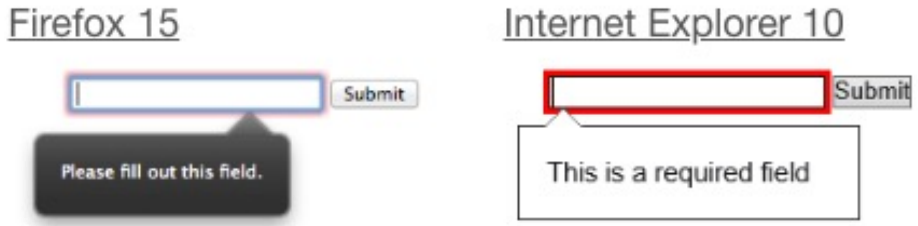
CSS Hooks

```
<form>
  <input type="text" id="foo" required />
  <input type="text" id="bar" />
</form>

<script>
  document.querySelectorAll
    ('input[type="text"]:invalid'); // matches input#foo
  document.querySelectorAll
    ('input[type="text"]:valid'); // matches input#bar
</script>
```

CSS Hooks

```
<style>
   :invalid { border: 1px solid red; }
   :valid    { border: 1px solid green; }
</style>
```



Removing Default Bubble

- The bubbles are the only means by which web browsers indicate the errors.
- We can apply a custom look of the bubbles across all supporting browsers.
- The only then option is to suppress default bubble and implement our own.
- If we do this, we must show error messages to users after invalid form submissions.

Removing Default Bubble

For example, this code will disable the default inline validation bubbles from all forms on a page.

```
var forms = document.getElementsByTagName('form');
for (var i = 0; i < forms.length; i++) {
    forms[i].addEventListener('invalid', function(e) {
        e.preventDefault();
        // display error messages to user here
    }, true);
}
```

Validitation API Limitations

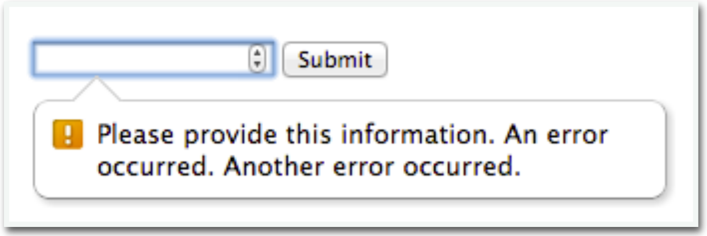
Problem #1: Handling multiple errors on one field.

- Calling setCustomValidity() on a node simply overrides its validationMessage.
- If we call this method on the same node twice the second call will overwrite the first.
- There is no mechanism to handle for an array of error messages.
- There isn't a way of displaying multiple error messages to the user.

Validitation API Limitations

- We can append additional error messages to the node's validationMessage.
- We cannot pass in HTML or formatting strings so we have to concatenating simple strings.

```
var foo = document.getElementById('foo');
foo.setCustomValidity
    (foo.validationMessage + ' An error occurred');
```



Please provide this information. An error occurred. Another error occurred.

Validitation API Limitations

Problem #2: Knowing when to check the validity of an input field.

- Consider the example of a form with two password input fields that must match.
- Whenever the value of either password field is changed the validity will be reevaluated.
- We need a means of running code whenever a field's validity might have changed.
- We can use the change event to implement this type of validation.

Validitation API Limitations

```
<form>
  <h2>Change Password</h2>
  <ul>
   <li><label for="pwd1">Password 1:</label>
   <input type="password" required id="pwd1" /></li>
   <li><label for="pwd2">Password 2:</label>
   <input type="password" required id="pwd2" /></li>
  </ul>
  <input type="submit" />
</form>
```

Validitation API Limitations

```
var password1 = document.getElementById('pwd1');
var password2 = document.getElementById('pwd2');
var checkPasswordValidity = function() {
  if (password1.value != password2.value)
    password1.setCustomValidity('Must match!'); // invalid
  else
    password1.setCustomValidity('');  // valid
};
password1.addEventListener
   ('change', checkPasswordValidity, false);
password2.addEventListener
   ('change', checkPasswordValidity, false);
```

Validitation API Limitations

Problem #3: Knowing when a user attempts to submit a form (contains valid or invalid data).

- The submit event is not fired until after the browser has determined the form is valid.
- It's useful to know when a user attempts to submit a form and it is prevented by the browser occurs.
- We may want to show the user a list of error messages, change focus, or display help text.

Validitation API Limitations

- We can add the novalidate attribute to the form and use its submit event.
- The form submission will not be prevented regardless of the validity of the data.
- We have to explicitly check whether the form contains valid data in a submit event and prevent submission accordingly.

Validitation API Limitations

```
<form id="passwordForm" novalidate>
    <h2>Change Password</h2>
    <ul>
      <li><label for="pwd1">Password 1:</label>
      <input type="password" required id="pwd1"/></li>
       <li><label for="pwd2">Password 2:</label>
       <input type="password" required id="pwd2"/></li>
    </ul>
    <input type="submit" />
</form>
```

Validitation API Limitations

```
var password1 = document.getElementById('pwd1');
var password2 = document.getElementById('pwd2');

var checkPasswordValidity = function() {
    if (password1.value != password2.value)
        password1.setCustomValidity('Must match!'); // invalid
    else
        password1.setCustomValidity('');  // valid
};
password1.addEventListener // validate field
    ('change', checkPasswordValidity, false);
password2.addEventListener // validate field
    ('change', checkPasswordValidity, false);
```

Validitation API Limitations

```
var form = document.getElementById('passwordForm');
form.addEventListener('submit', function() {
    checkPasswordValidity(); // validate form
    if (!this.checkValidity()) {
        event.preventDefault();
        // display error messages to user here
        password1.focus();
    }
}, false);
```

Validitation API Limitations

- Adding the novalidate attribute to a web form prevents the browser from displaying the inline validation bubble to the user.
- We must implement our own means of presenting error messages to the user.
- We need a forminvalid event that would be fired whenever a form submission was prevented due to invalid data.

Validitation API Limitations

```
<form id="passwordForm"  novalidate>
    <h2>Change Password</h2>
    <ul>
      <li><label for="password1">Password 1:</label>
      <input type="password" required id="password1"/>
      <p class="error"></p></li>
      <li><label for="password2">Password 2:</label>
      <input type="password" required id="password2"/></li>
    </ul>
    <input type="submit" />
</form>
```

Validitation API Limitations

```
.error {
    display: none; color: red; font-weight: bold;
}

.submitted :invalid + .error { display: block; }
.submitted :invalid { border: 1px solid red; }
```

Validitation API Limitations

```
var checkPasswordValidity = function() {
    if (password1.value != password2.value) {
        password1.setCustomValidity('Must match!'); // invalid
     } else password1.setCustomValidity(''); // valid
};

var updateErrorMessage = function() {
    form.getElementsByClassName('error')[0]
        .innerHTML = password1.validationMessage;
};
```

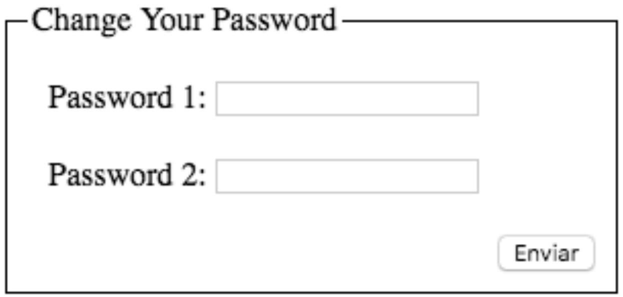
Validitation API Limitations

```
var password1=document.getElementById('password1');
var password2=document.getElementById('password2');
var form = document.getElementById('passwordForm');
password1.addEventListener
    ('change', checkPasswordValidity, false);
password2.addEventListener
    ('change', checkPasswordValidity, false);
```

Validitation API Limitations

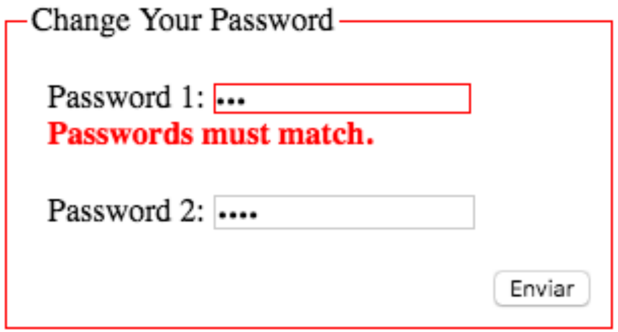
```
form.addEventListener('submit', function(event) {
    if (form.classList) form.classList.add('submitted');
    checkPasswordValidity();
    if (!this.checkValidity()) {
        event.preventDefault();
        updateErrorMessage();
        password1.focus();
    }
}, false);
```

Validitation API Limitations


Change Your Password

Password 1:

Password 2:


Change Your Password

Password 1: •••

Passwords must match.

Password 2: ••••

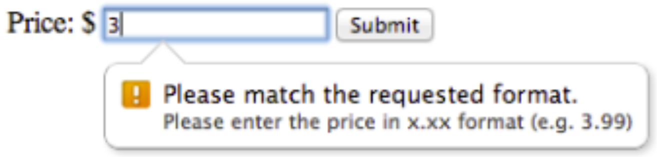
Title Attribute

While it doesn't change the validationMessage, browsers display the contents of the title attribute in the inline bubble (if it's provided).

```
<form>
   <label for="price">Price: $</label>
   <input type="text" pattern="[0-9].[0-9][0-9]"
      title="Enter price in x.xx format (e.g. 3.99)"
      id="price" value="3" />
   <input type="submit" value="Submit" />
</form>
```

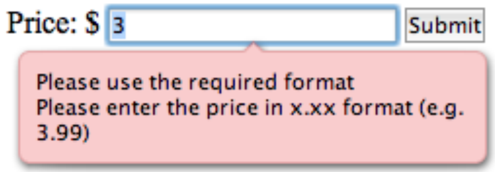
Title Attribute

Chrome 21



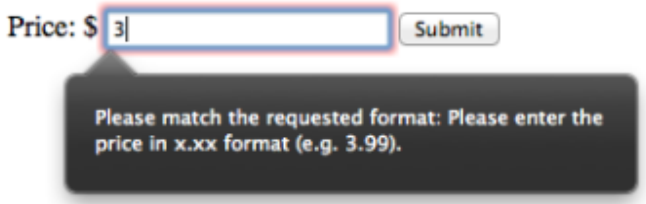
Please match the requested format.
Please enter the price in x.xx format (e.g. 3.99)

Opera 12



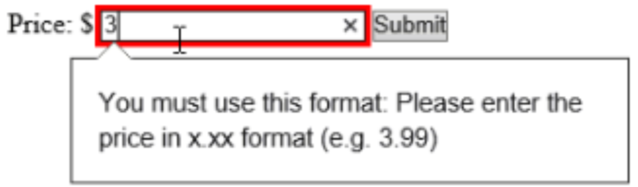
Please use the required format
Please enter the price in x.xx format (e.g. 3.99)

Firefox 15



Please match the requested format: Please enter the price in x.xx format (e.g. 3.99).

IE 10



You must use this format: Please enter the price in x.xx format (e.g. 3.99)

Title Attribute

- The best time to give user feedback is after they interact with a field, not before.
- We can add a class to the input fields after they have been interacted with and only apply the borders when the class is present.

Title Attribute

```
<style>
   .interacted:invalid { border: 1px solid red; }
   .interacted:valid   { border: 1px solid green; }
</style>
<form>
   <input type="text" required />
   <input type="text" />
   <input type="submit" />
</form>
```

Title Attribute

```
<script>
  var inputs=document.querySelectorAll('input[type=text]');
  for (var i = 0; i < inputs.length; i++) {
    inputs[i].addEventListener('blur', function(event) {
      event.target.classList.add('interacted');
    }, false);
  }
</script>
```