/01

Using HTML5 Form Validation

L. Hernández | 2023

Using HTML5 Form Validation

Imagine, for example, that we want to create an *HTML form* that includes a *form field* for entering a *Social Security* number.

In that case, we can use the **required** *attribute* to ensure that a value has been entered.

Then, we can use the **pattern** *attribute* to ensure that entered value matches the pattern (for a valid *Social Security* number).

This *HTML form* illustrates how we can use the **required** *attribute*:

```
<form>
  <div>
    <label>
      Social Security Number:
      <input id="ssn" required />
    </label>
  </div>
  <div>
    <input type="submit" />
  </div>
</form>
```

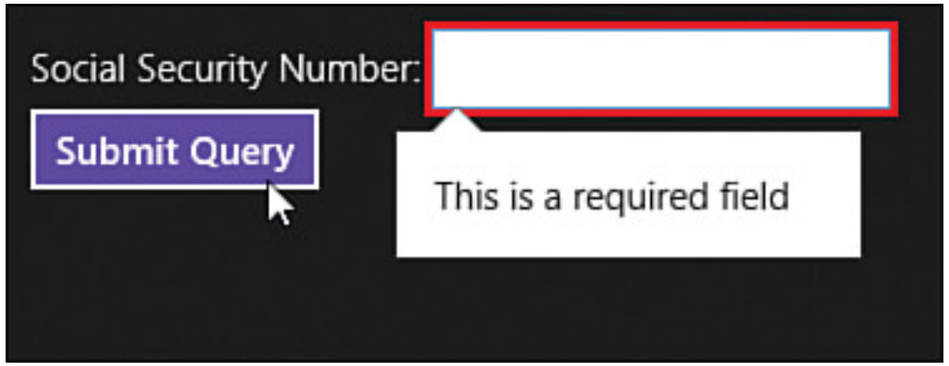
Using **required** Attribute

/03

/04

If we submit this *form*, and we do not enter a value for the **ssn** field, then we get an *error message*.

The **input** *element* will have a red border surrounding it and a callout message is displayed.



Using **required** Attribute

Using **pattern** Attribute

We use **pattern** *attribute* to validate a value entered into an *input field* against a *regular expression pattern*.

The **pattern** *attribute* is not triggered unless we enter a value.

This field includes a **title** *attribute* that contains the *format* displayed by the *pattern error message*.

This HTML form will validate the *Social Security* number against a *regular expression*:

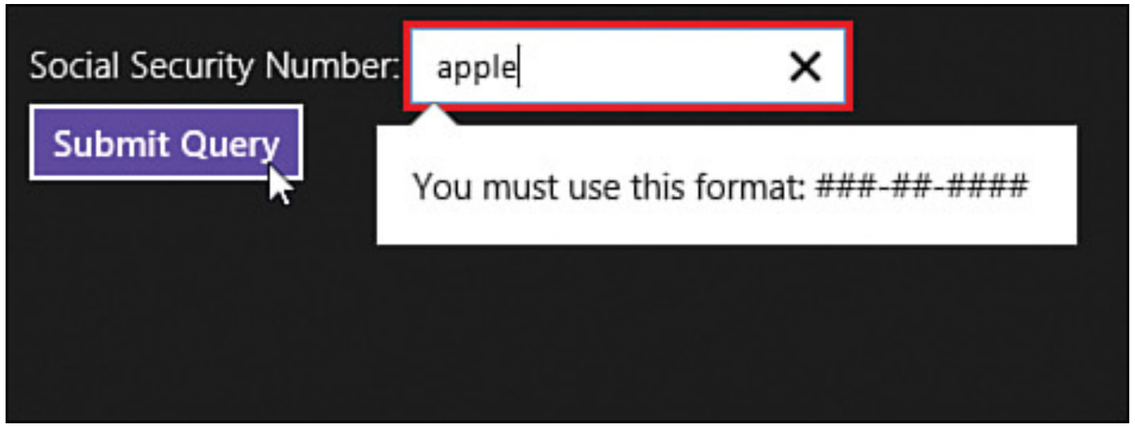
```
<form>
  <div>
    <label>
      Social Security Number:
      <input id="ssn"
        required
        pattern="^\d{3}-\d{2}-\d{4}$"
        title="###-##-####" />
    </label>
  </div>
  <div><input type="submit" /></div>
</form>
```

Using **pattern** Attribute

/06

/07

If we enter an invalid *Social Security* number then we get a *validation error message*.



Using **pattern** Attribute

Performing Custom Validation

If we need to add *custom validation rules* to a **form** *element* then we could use JavaScript **setCustomValidity()** method.

We can use this method to associate a *custom validation error message* with a *form field*.

Performing Custom Validation

Imagine that we have a complex set of rules for validating a username in a *user registration form*.

We want to ensure that username is a certain length, unique in the database, and does not contain special characters.

This *HTML form* includes a **userName** identifier in its *input field*:

```
<form>
  <div>
    <label>
      User Name:
      <input id="userName" required />
    </label>
  </div>
  <div><input type="submit" /></div>
</form>
```

Performing Custom Validation

Performing Custom Validation

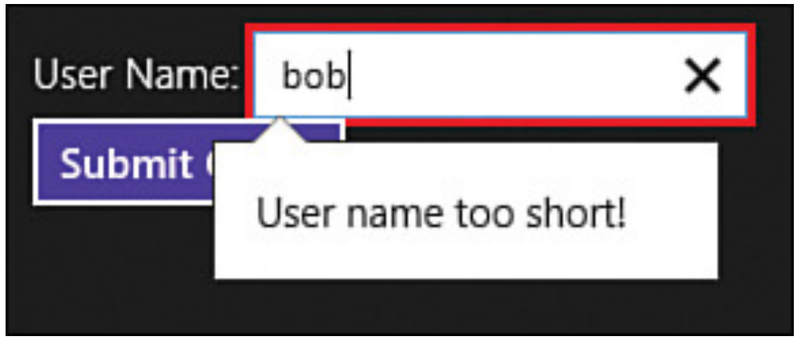
This JavaScript code demonstrates how we can display a *validation error message* when the user name is too short.

```
var userName = document.getElementById("userName")
userName.addEventListener("input", function (evt) {
  // User name must be more than 3 characters
  if (userName.value.length < 4) {
    userName.setCustomValidity("User name too short!");
  } else {
    userName.setCustomValidity(""); // clear error
  }
});
```

Performing Custom Validation

- In this example code, an *event listener* for the **input** *event* is first created.
- When the value of the **input** *element* is changed then the length of the value is checked.
- If the username is less than four characters then the **setCustomValidity()** method is used to *invalidate* the **input** *element*.
- Otherwise, the **setCustomValidity()** method is called with an empty string to clear any previous *validation errors* associated with the **input** *element* (**userName**).

After we submit the *form* in this example, we must see a *validation error message*.



Performing Custom Validation

Performing Custom Validation

The **input** *event* is raised as soon as the contents of an **input** *element* are changed.

However, the **change** *event* is not raised until after the **input** *element* loses focus.

Customizing Validation Error Style

By default, *invalid HTML fields* in a *form* appear with a red border.

For example, if we submit an *HTML form* without entering a value in a *required field*.

Customizing Validation Error Style

We can customize appearance of *form fields* in different *states of validity* by using *CSS pseudo classes*.

:valid—Applies when an **input** *element* is valid

:invalid—Applies when an **input** *element* is invalid

:required—Applies when an **input** *element* is required (has the **required** *attribute*)

:optional—Applies when an **input** *element* is not required (does not have the **required** *attribute*)

Imagine that we have created this *user registration form* and we want to set *states of validity* by using *CSS pseudo classes*.

```
<form>
  <div>
    <label>
      First Name:
      <input id="firstName" required />
    </label>
  </div>
  ...
  <div>
    <label>
      Company: <input id="company" />
    </label>
  </div>
  <div><input type="submit" /></div>
</form>
```

Customizing Validation Error Style

Customizing Validation Error Style

This *form* contains *required fields* for the user first and last names but it also contains an optional field for the user company.

We could then use these *style rules* to control how the **input** *elements* are styled:

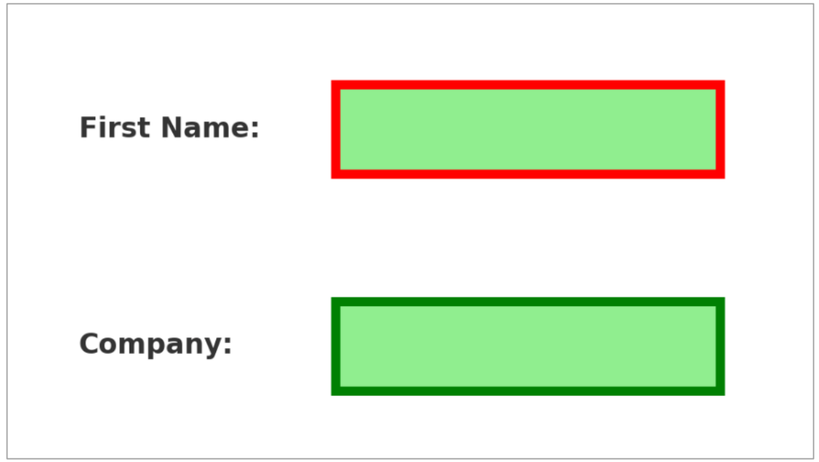
```
:valid {
   background-color: green;
}
:invalid {
   background-color: yellow;
}
:optional {
   border: 4px solid green;
}
:required {
   border: 4px solid red;
}
```

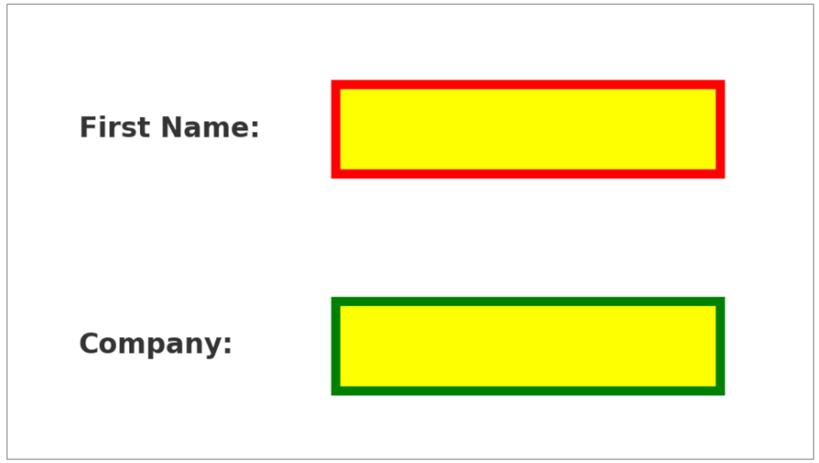
Customizing Validation Error Style

Customizing Validation Error Style

These *style rules* cause *valid fields* to appear with a green background color and *invalid fields* to appear with a yellow background color.

Optional fields appear with a green border and *required fields* appear with a red border.





After we successfully submit a *form*, we need to reset it so we can use this *form* again.

If we are using *validation attributes* then we cannot reset a *form* simply by assigning *empty strings* to the *form fields*.

If we assign an *empty string* to a *required field* then this field will be in an *invalid state*.

Resetting a Form

Resetting a Form

```
<form id="frmAdd">
  <div>
    <label>
      Title:
      <input id="inpTitle" required />
    </label>
  </div>
  <div><input type="submit" /></div>
</form>
```

Instead, we should reset a *form* by calling the JavaScript **reset()** method. This method throws a *form* back into its *default state*.

For example, here's a form for adding new movies. Each time we add a movie, we want the *form* to reset to its *default state*.

Here's the JavaScript code that we can use for handling the *form submit event* in this case.

```
function initialize() {
  var frmAdd = document.getElementById("frmAdd");
  var inpTitle = document.getElementById("inpTitle");

  frmAdd.addEventListener("submit", function (evt) {
    evt.preventDefault();
    var newMovie = {
      title: document.getElementById("inpTitle").value
    };
    addMovieToDb(newMovie).done(function () {
      frmAdd.reset();
    });
  });
  ...
}
...
```

Resetting a Form

Resetting a Form

This code will add the movie title to a *database* and then resets the *form* so the form returns to its *default state*.

Here, the **reset()** method is used to return the *form* to its *default state*.

Resetting a Form

If we neglect to call **evt.preventDefault()** in the *form submit handler* then page will be submitted and reloaded.

We don't want to do this and we want to avoid ever submitting back to the server.

- **search**
- **tel**
- **url**
- **email**
- **datetime**
- **date**
- **month**
- **week**
- **time**
- **datetime-local**
- **number**
- **range**
- **color**

Using HTML5 Input Elements

There are many *standard input element types* such as **<input type="text" />** and **<input type="checkbox" />**.

Different *input types can* have different *appearances* and accept different *types of data*.

HTML5 adds several new *input types* but not all their *features forms* are supported by all browsers.

Using HTML5 Input Elements

We can take advantage of new *HTML5 input types* to enforce *validation rules*.

For example, an **<input type="number" />** will accept only numerals and not other types of characters.

We also can take advantage of new *HTML5 input types* to control the *user interface* for entering a value into a field.

For example, when we're using a *touch keyboard*, an **<input type="email" />** field displays a *specialized keyboard for entering email addresses*, which includes *specialized keys* such as "@" and ".com".

Using HTML5 Input Elements

Labeling Form Fields

We need to know the proper way to label *HTML form fields*.

Providing proper labels is very important for making web pages accessible to users.

Labeling Form Fields

There are two ways that we can use a *label element* to label a *form element*.

If we want the label to appear right next to the *form element* then we can include this element inside the *label's opening and closing tags*.

```
<label>
 Title:
 <input id="inpTitle" required />
</label>
```

Labeling Form Fields

If a label is separated *from element* being labeled in a page then we can associate *label element* and *form element* explicitly by using the *label's* **for** *attribute*.

```
<label for="inpTitle">
  Title:
</label>
// Other Content
<input id="inpTitle" required />
```

Labeling Form Fields

If we need to provide *additional hints* about the appropriate **input** value for a *form element* then we can take advantage of a new HTML5 *attribute*.

When used in any *form*, the **placeholder** *attribute* creates a *watermark*.

This *HTML form* contains a *form field* for entering a *product activation code*:

```
<label>
  Activation Code:
  <input id="activationCode"
    size="10"
    placeholder="##-####-##" />
</label>
```

Labeling Form Fields

Labeling Form Fields

This *form* includes a **placeholder** *attribute* which displays the text "##-####-##".

As soon as we start typing a value into this field, the *placeholder text* disappears.

Entering a Number

If we want to prevent a *user from* entering anything except a number into an *input field* then we should use the **type="number"** *attribute*:

```
<label>
  Favorite Number:
  <input id="inpFavNumber"
    type="number"
    placeholder="###" />
</label>
```

Entering a Number

When we enter anything that is not a number in this *input type*, then the value disappears as soon as the field loses focus.

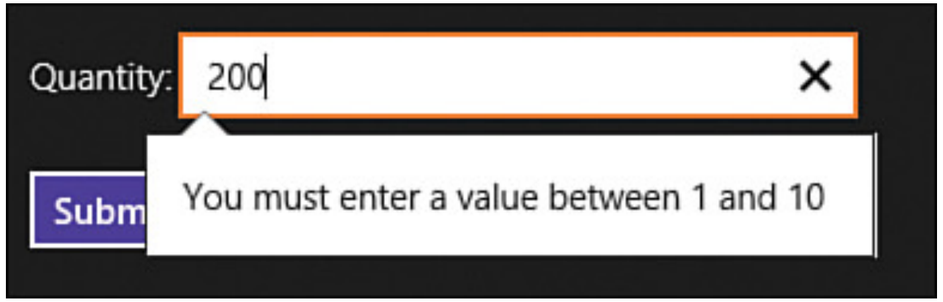
Because this can be confusing to the user, we could include a **placeholder** *attribute* that indicates that the field only accepts numbers.

We can use the **min** and **max** *attributes* to specify a *minimum and maximum value* for the *input field* when we're using the **type="number"** *attribute*.

```
<label>
  Quantity:
  <input id="inpQuantity"
    type="number"
    min="1"
    max="10"
    placeholder="###" />
</label>
```

Entering a Number

If we enter a number that does not fall into *specified range* then a *validation error message* will be displayed.



Entering a Number

When we use a *number field* and we are using a *touch keyboard*, we will get a *special keyboard* for entering numbers automatically.

Entering a Number

Entering a Number

The **step** *attribute* determines the allowable increment between numbers.

Entering a Number

By default, we can only enter an *integer value* into an **<input type="number" />** *field*.

If we want to enter a *non-integer value*, such as **1.5**, then we need to modify the **step** *attribute*:

```
<input id="inpFavNumber"
  type="number"
  step="0.5"
  placeholder="###" />
```

If we want to display a *slider*, then we can create an **<input type="range" />** *element*.

For example, this *HTML form* will display a *slider* that enables us to select a quantity of candy to buy.

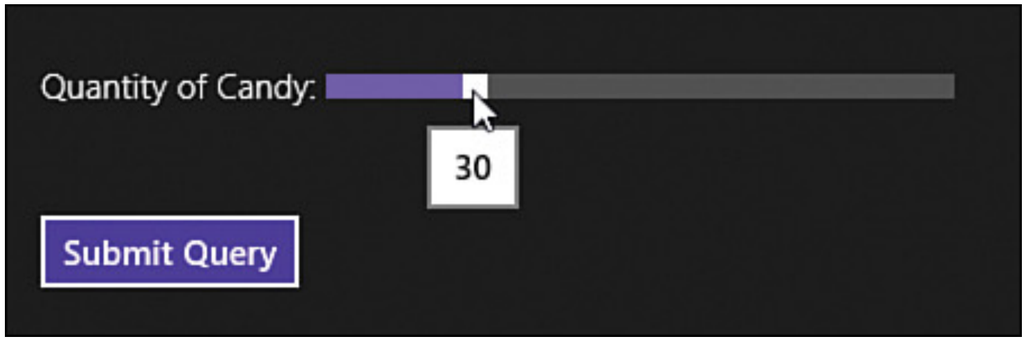
```
<label>
  Quantity of Candy:
  <input id="quantity"
    type="range"
    min="10"
    max="100"
    step="5"
    value="30"/>
</label>
```

Entering a Value from a Range of Values

Entering a Value from a Range of Values

This *slider* displays a range of values between **10** and **100** with ***5**-unit* increments.

The default value is set to **30**.

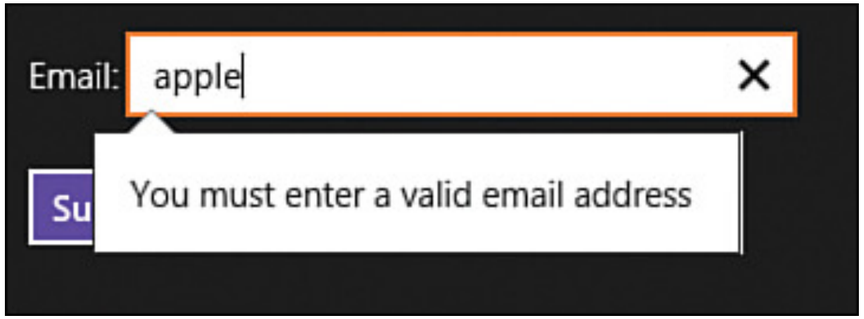


Using Others HTML5 Inputs Types

We can also use the *input types* **email**, **url**, **tel**, and **search** to enable users to enter email addresses, URLs, telephone numbers, and search terms.

Using **<input type="email" />** *field* also gives us *automatic validation*.

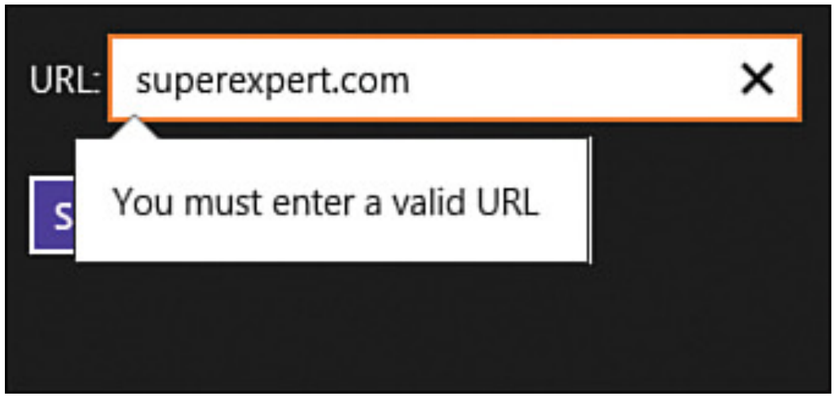
In this case, we should enter a *valid email address* or we'll get a *validation error message*.



Using Others HTML5 Inputs Types

An **<input type="url" />** *field* requires we to enter a *valid absolute URL*.

For example, **http://Superexpert.com** and **ftp://Superexpert.com** are valid URLs, but **superexpert.com** and **www.superexpert.com** are not because they are not *absolute URLs*.



Using Others HTML5 Inputs Types

Using Others HTML5 Inputs Types

Using **<input type="url" />** creates a special *input field* for entering URLs.

In this case, we'll get a *touch keyboard* which includes special "/" and ".com" keys.

Using Others HTML5 Inputs Types

We can use the **<input type="tel" />** *element* to enter *telephone numbers*.

Because there are so many different formats for them, this *input type* does not perform any *validation*.

However, we can use it to display a *specialized touch keyboard* for *telephone numbers*.

Using Others HTML5 Inputs Types

There is also an **<input type="search" />** *element* and this behaves identically to any **<input type="text" />** *element*.

The only difference is that we get a *custom keyboard* with a "Search" key instead of an "Enter" key.

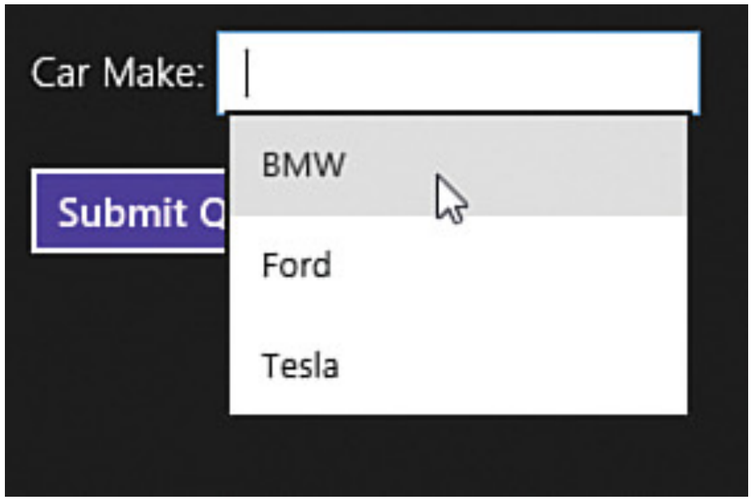
We can use the new *HTML5* **list** *attribute* to provide an *auto-complete experience* for users.

For example, this code provides a *list of suggestions* for our car make:

```
<label>
  Car Make:
  <input id="inpCarMake"
    list="dlCarMakes" />
  <datalist id="dlCarMakes">
    <option>BMW</option>
    <option>Ford</option>
    <option>Tesla</option>
  </datalist>
</label>
```

The **list** *attribute* points at an *HTML5 datalist element* which contains the *list of suggestions*.

When we start entering text into this *input element* then we get the *suggestions*.



Entering a Value from a List of Values

Entering a Value from a List of Values

We are not forced to select a option from the list.

Using the **list** *attribute* makes an **input** *element* work more like a *combo box* than a *select list*.

Selecting Files

We can use **<input type="file" />** to create a *file picker*.

For example, we can use it to enable a user to select a *picture file* from their file system.

This *HTML code* includes an **<input type="file" />** *element* and a **div** *element* with an **id** *attribute*.

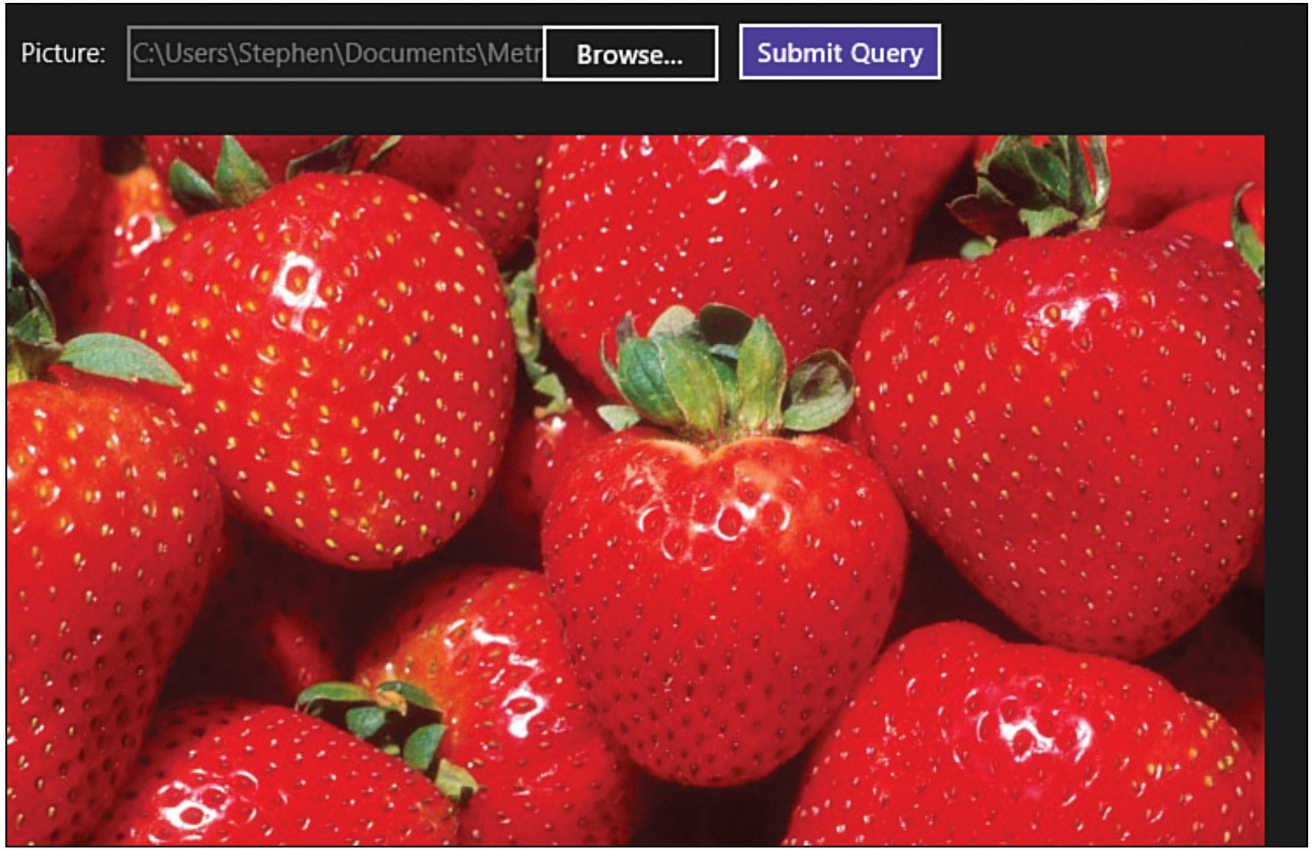
```
<form id="frmAdd">
  <div>
    <label>
      Picture:
      <input id="inpFile" type="file" accept="image/*" />
    </label>
    <input type="submit" />
  </div>
</form>

<img id="imgPicture" />
```

Selecting Files

Selecting Files

After we select a *picture* from our file system, this *picture* will appear in the **img** *element*.



Selecting Files

When we use the **<input type="file" />** *element*, we can use the **accept** *attribute* to restrict *type of files* that can be uploaded.

For example, in preceding *HTML markup*, this **accept** *element* has the value **"image/*"**, which prevents any file except image files from being selected.

Selecting Files

The following JavaScript code is used to handle the *form submit event*.

This code grabs the *selected picture file* from the **input** *elements files collection* and then displays it in the **img** *element*.

```
function init() {
  var frmAdd = document.getElementById("frmAdd");
  frmAdd.addEventListener("submit", function (evt) {
    evt.preventDefault();
    var imgPicture =
      document.getElementById("imgPicture");
    var inpFile = document.getElementById("inpFile");
    if (inpFile.files.length > 0) {
      // Use HTML5 File API to create object URL
      // to refer to photo file
      var pictureUrl =
        URL.createObjectURL(inpFile.files[0]);
      // Show photo in IMG element
      imgPicture.src = pictureUrl;
    }
  });
}

document.addEventListener("DOMContentLoaded", init);
```