

HTML5 Training for Web Developers

HTML5 Web Storage

Lesson 1, Activity 2: Overview of HTML5 Web Storage

Before HTML5, the standard way of storing data on the client was to use cookies. But cookies weren't really meant to store a lot of data and the JavaScript methods for accessing, modifying and removing them are a bit clumsy.

HTML5 offers a very simple and straightforward way to create, read, update, and delete variables that are stored between user sessions (`localStorage`) or just for a single session (`sessionStorage`). We'll take a look at both of these.

Lesson 1, Activity 3: Web Storage

Before HTML5, the standard way of storing data on the client was to use cookies. But cookies weren't really meant to store a lot of data and the JavaScript methods for accessing, modifying and removing them are a bit clumsy.

HTML5 offers a very simple and straightforward way to create, read, update, and delete variables that are stored between user sessions (`localStorage`) or just for a single session (`sessionStorage`). We'll take a look at both of these.

HTML5 Web storage is used to store key-value pair data throughout a single session (`sessionStorage`) or between sessions (`localStorage`). Browsers typically limit the amount of client-side storage space allocated to a single domain to 5 megabytes and throw a `QUOTA_EXCEEDED_ERR` exception if you try to store more than that.

The methods and properties are the same for working with `sessionStorage` and `localStorage` and are shown below:

Web Storage Methods and Properties	
Method/Property	Description
length	Holds the number of key/value pairs.
setItem(key,value)	Creates or updates a key/value pair.
getItem(key)	Gets the value of the specified key.
key(n)	Returns the nth key. Useful for iterating through key/value pairs.
removeItem(key)	Removes the key/value pair for the given key.
clear()	Removes all key/value pairs.

Browser Support

The following browsers support Web Storage:

- 1. Internet Explorer 8+
- 2. Firefox 3.5+
- 3. Chrome 4.0+
- 4. Safari 4.0+
- 5. Opera 10.5+

Internet Explorer and Firefox do not support web storage unless the files are delivered by a web server. So unless you have a web server (e.g., IIS or Apache) set up locally, you should use Safari, Chrome, or Opera for the demos and exercises in this lesson.

Local Storage

Local storage is used for maintaining state during and between sessions. It is easiest to understand through a simple example:

- 1. Open [html5-storage/Demos/local-storage.html](#) in your browser:

Key: Value:

#	key	value	Delete
Nothing to Show			

- 2. Enter a key/value pair and press the **Save** button. The table should update:

Key: NY Value: New York

#	key	value	Delete
0	NY	New York	<input type="button" value="X"/>

- 3. Add some more key/value pairs:

Key: CA Value: Caifornia

#	key	value	Delete
0	CA	Caifornia	<input type="button" value="X"/>
1	TX	Texas	<input type="button" value="X"/>
2	NY	New York	<input type="button" value="X"/>

- 4. Refresh your browser. The values you entered should still be in the table.
- 5. Close and reopen your browser. The values you entered should **still** be in the table.
- 6. Enter a key that you have already entered, but change the value. Press **Save**. The table should update:

Key: CA Value: Caifornia

#	key	value	Delete
0	CA	Caifornia	<input type="button" value="X"/>
1	TX	Texas	<input type="button" value="X"/>
2	NY	New York	<input type="button" value="X"/>

- 7. Press the **Delete** button next to one of your key/value pairs. It should (after you bypass the warning) be removed from the table and stay removed if you refresh or close and open your browser.
- 8. Press the **Clear All** button below the table. All key/value pairs should (after you bypass the warning) be removed from the table and stay removed if you refresh or close and open your browser.

Let's take a look at the code starting with the HTML in the :

Key:

Value:

#	key	value	Delete
Nothing to Show			

Clear All

Things to notice:

1. We will use the `ids` of the `input` elements to get the user-entered values.
2. The **Save** button calls the `save()` function.
3. The **Clear All** button calls the `clearStorage()` function.
4. We will update the rows in the `tbody` element to show the key/value pairs in `localStorage`.

Now let's look at the JavaScript, starting with the `updateTable()` function, which is called when the page loads and each time we set or remove an item from `localStorage`:

```

window.addEventListener("load",updateTable,false);

function updateTable() {
  var tbody = document.getElementById("output");
  while (tbody.getElementsByTagName("tr").length > 0) {
    tbody.deleteRow(0);
  }
  var row;
  if (localStorage.length==0) {
    row = tbody.insertRow(0);
    cell = row.insertCell(0);
    cell.colspan="4";
    cell.innerHTML = "Nothing to Show";
  }
  for (var i=0; i < localStorage.length; ++i) {
    row = tbody.insertRow(i);
    cell = row.insertCell(0);
    cell.innerHTML = i;
    cell = row.insertCell(1);
    cell.innerHTML = localStorage.key(i)
    cell = row.insertCell(2);
    cell.innerHTML = localStorage.getItem(localStorage.key(i));
    cell = row.insertCell(3);
    cell.innerHTML = "
  }
};
}
}

```

Things to notice:

1. We use the standard [table element methods](#) to remove all the table rows in the `tbody`.
2. If the `localStorage` is empty (`localStorage.length==0`) then we create a single row that reads "Nothing to Show".
3. If the `localStorage` is not empty, we loop through it and create a row for each key/value pair.
4. Note that when the **delete** image is clicked, the `deleteItem()` function is called and the key for that key/value pair is passed in.

Now let's look at the remaining functions:

```

function deleteItem(key) {
  if (!confirm("Are you sure you want to delete this item?")) return;
  localStorage.removeItem(key);
  updateTable();
}

function clearStorage() {
  if (!confirm("Are you sure you want to delete all local storage for this domain?"))
    localStorage.clear();
  updateTable();
}

function save() {
  var key = document.getElementById("key").value;
  var value = document.getElementById("value").value;
  localStorage.setItem(key,value);
  updateTable();
}

```

Things to notice:

1. The `deleteItem()` function uses `localStorage.removeItem(key)` to remove the passed-in key.
2. The `clearStorage()` function uses `localStorage.clear()` to empty `localStorage`.
3. The `save()` function uses `localStorage.setItem(key, value)` to add or update a key/value pair.
4. All three functions call `updateTable()` after they run so that the page gets updated.

And that's all there is to `localStorage`.

Use Case

A good use case for `localStorage` is saving data in a large form (for example - a job application) to finish and submit later. This way, you don't have to clog up your database server with half-completed applications.

Session Storage

And here's the beauty of it: `sessionStorage` is exactly the same. You can literally use `find` and `replace` to change "localStorage" to "sessionStorage" and your page will continue to work. The only difference is that your key/value pairs will only be accessible for the life of the session. So, if you close and reopen your browser, your key/value pairs will be gone.

To test this, go through the same [process](#) we went through to test `localStorage`, but use [html5storage/Demos/session-storage.html](#) instead.

Browsers are free to maintain a session longer than a single visit. For example, if the browser crashes, the browser may try to restore the session, in which case it may be able to keep the key/value pairs in `sessionStorage`.

Use Case

A good use case for `sessionStorage` is auto-saving forms so that if a user accidentally refreshes the browser, he/she doesn't lose any data. We'll tackle this in the upcoming exercise.

Prefixing your Keys

As it's possible to have multiple applications fed from the same origin (think URL), it's a good idea to use prefixes to scope your keys to a specific application. Take a look at the following demo:

Code Sample:<html5-storage/Demos/questions.html>

```
---- CODE OMITTED ----
```

Question:

Answer:

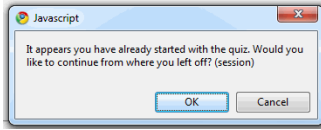
This is very similar to the page we saw earlier. It just prepends a prefix to the key, so that keys used in other applications fed from the same domain don't get overwritten. It also adds a convenient feature of emptying the question and answer and replacing the cursor in the question.

Lesson 1, Activity 5: Creating a Quiz Application

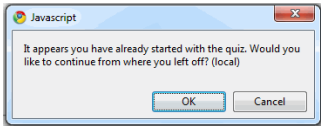
Duration: 30 to 45 minutes.

In this exercise, you will create a quiz application that allows the user to save and resume later. It also protects the user from losing data if he/she accidentally refreshes.

1. Open <html5-storage/Solutions/saving-quiz.html> in your browser and play with the application:
 1. Answer one or more questions and then refresh the browser. You will get a dialog giving you the chance to refill the form with values stored in `sessionStorage`:



2. Press **OK** and your values get added back.
3. Refresh again and press **Cancel** on the dialog. Your values do not get added back.
4. Refresh again. You don't get the JavaScript dialog because the `sessionStorage` key/value pairs were removed when you pressed **Cancel** in the previous step.
5. Answer one or more questions again and click the **Save My Answers for Later** button.
6. Close and reopen the browser. You will get a dialog giving you the chance to refill the form with values stored in `localStorage`:



7. If you click **Cancel**, the `localStorage` key/value pairs will be removed and you'll have to start the quiz over.
8. If you click **OK**, the form will be refilled with your previous answers and they will be saved into `sessionStorage`.
9. Also notice that the footer contains the time and date the quiz answers were last saved:

Answers last saved: 2010-12-15 17:36:23

This is the challenge to the exercise.

2. Now open <html5-storage/Exercises/saving-quiz.html> in your editor.
3. In the `addLoadEvents()` function:
 1. Loop through the inputs and add event listeners that capture change events to save the associated key/value pair in `sessionStorage`. Don't forget to use the prefix.
 2. Add an event listener to the **Save** button to capture a click event and call `saveAnswers`.
 3. Call the `refill()` function at the end.
4. Write the code in the `saveAnswers()` function to save all the answers in `localStorage`.
5. The `refill()` function:
 1. calls `hasAnswers()`, which returns "session", "local", or false, depending on if and where it finds saved answers. If there are no saved answers, it returns without doing anything.
 2. declares some variables:
 1. `confirmed` - we'll change it to true if the user wishes to refill the form.
 2. `msg` - the message to ask the user if he/she wants to refill the form.
 3. `questions` - the question inputs
 3. loops through the inputs. On the first iteration, it prompts the user with the message. If the user clicks **Cancel**, the key/value pairs are deleted (via the `deleteAnswers()` function) and the function returns/ends. Otherwise, we iterate through the questions. This is where you come in...
 4. Add code to populate the question inputs from the appropriate storage location (based on the value of `fillFrom`).

Challenge

Notice that an external script called `dateFormat.js` is included. That extends the `Date` object prototype with a `format()` method, which you use as follows:

```
var now = new Date();
var dateMask = "yyyy-mm-dd H:MM:ss";
var formattedNow = now.format(dateMask);
```

Use this to write out the date last saved to the output element below the form. You will need to store the date in `sessionStorage` and `localStorage` as appropriate. Don't forget the prefix. Note that the `dateMask` variable is already set in the code.

Solution:

<html5-storage/Solutions/saving-quiz.html>

```
<!DOCTYPE HTML>
---- C O D E   O M I T T E D ----

function addLoadEvents() {
  document.getElementById("quiz").addEventListener("change",function() {
    updateMeasures();
  }, false);
  var questions = document.getElementById("quiz").getElementsByTagName("input");
  for (var i=0; i < questions.length; ++i) {
    questions[i].addEventListener("change",function() {
      sessionStorage.setItem(prefix+this.id,this.value);
    },false);
  }
  document.getElementById("save").addEventListener("click",saveAnswers,false);
  refill();
}

function saveAnswers() {
  var questions = document.getElementById("quiz").getElementsByTagName("input");
  for (var i=0; i < questions.length; ++i) {
    localStorage.setItem(prefix+questions[i].id,questions[i].value);
  }
}

function refill() {
  /*
   * If any answers were saved (local or session), then let's fill the form with the saved answers.
   * hasAnswers() returns either "local", "session", or false
   */
  var fillFrom = hasAnswers();

  //If no answers were saved, we can't refill the form so let's just get out of the function with "return"
  if (!fillFrom) return;
  /*
   * If we get to this point, then we know we have answers saved that will be used to refill the form
   * var "confirmed" will be used to have the user confirm the refilling of the answers (false by default)
   */
  var confirmed=false;
  //this is the message the user will see to confirm refilling the form
  var msg="It appears you have already started with the quiz. Would you like to continue from where you left off? (" + fillFrom + ")";
  //let's get all the questions
  var questions = document.getElementById("quiz").getElementsByTagName("input");
  //let's loop all the questions
```

```

for (var i=0; i < questions.length; ++i) {
  /*
   * if refilling is not confirm by the user, let's delete the answers and get out of this function
   * confirm() is a built-in JS method that displays a dialog box with the specified message, along with an OK and a Cancel button. The method returns true if the user clicks on "OK", else it returns false.
   */
  if (!confirmed && !confirm(msg)) {
    deleteAnswers();
    return;
  }
  /*
   * if we get to this point, then the user has confirmed refilling, let's set the var "confirmed" to true.
   */
  confirmed=true;
  //if we're refilling from the session...
  if (fillFrom == "session") {
    /*
     * we need the || "" for IE, which returns null if the key is not found.
     * for the current question in the loop's iteration, set the corresponding value from the session
     */
    questions[i].value=sessionStorage.getItem(prefix+questions[i].id) || "";
  }
  //else, we're refilling from the local storage...
  else {
    /*
     * again, we need the || "" for IE, which returns null if the key is not found.
     * for the current question in the loop's iteration, set the corresponding value from the local storage
     */
    questions[i].value=localStorage.getItem(prefix+questions[i].id) || "";

    //let's also save the answer to the session
    sessionStorage.setItem(prefix+questions[i].id,questions[i].value);
  }
  //after all the refilling is done, update the measures
  updateMeasures();
}
---- C O D E   O M I T T E D ----

```

Challenge Solution:
<html5-storage/Solutions/saving-quiz-challenge.html>

```

<!DOCTYPE HTML>
---- C O D E   O M I T T E D ----
var dateMask = "yyyy-mm-dd H:MM:ss";
function addLoadEvents() {
  document.getElementById('quiz').addEventListener("change",function() {
    updateMeasures();
  }, false);
  var questions = document.getElementById("quiz").getElementsByTagName("input");
  for (var i=0; i < questions.length; ++i) {
    questions[i].addEventListener("change",function() {
      var now=new Date();
      sessionStorage.setItem(prefix+this.id,this.value);
      document.getElementById("dateLastSaved").innerHTML=now.format(dateMask);
      sessionStorage.setItem(prefix+"dateLastSaved",now.format(dateMask));
    },false);
  }
  document.getElementById("save").addEventListener("click",saveAnswers,false);
  refill();
}

function saveAnswers() {
  var questions = document.getElementById("quiz").getElementsByTagName("input");
  var now=new Date();
  for (var i=0; i < questions.length; ++i) {
    localStorage.setItem(prefix+questions[i].id,questions[i].value);
  }
  localStorage.setItem(prefix+"dateLastSaved",now.format(dateMask));
}

function refill() {
  var fillFrom = hasAnswers();
  if (!fillFrom) return;
  var now;
  ---- C O D E   O M I T T E D ----
  if (fillFrom == "session") {
    now = sessionStorage.getItem(prefix+"dateLastSaved");
  } else {
    now = localStorage.getItem(prefix+"dateLastSaved");
    sessionStorage.setItem(prefix+"dateLastSaved",now);
  }

  document.getElementById("dateLastSaved").innerHTML=now;
  updateMeasures();
}
---- C O D E   O M I T T E D ----
<small>Answers last saved: <output id="dateLastSaved">not saved</output></small>
</body>
</html>

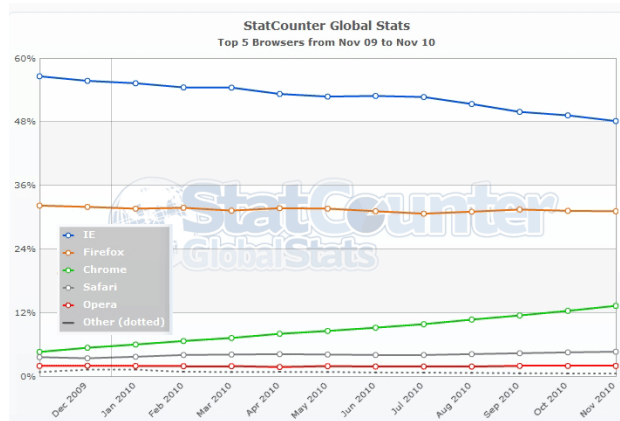
```

Lesson 1, Activity 6: Other Storage Methods

While HTML5 Web Storage makes it a lot simpler to store key/value pairs locally, it does not allow for the same kind of complex data storage that we get on the server via relational databases. One way to get a lot more out of the `sessionStorage` and `localStorage` mechanisms is to use object literals (JSON) to store complex data in a key/value pair. However, the brains behind HTML5 have been thinking bigger.

Web Database Storage

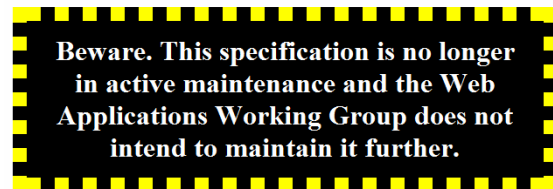
One promising idea, which has actually been implemented by several browsers, was to store data on the client in the same way we usually do it on the server: in a SQL database. The W3C [spiced it out under the name of Web Database Storage](#), and Chrome, Opera, and Safari implemented this with SQLite. But Mozilla and Microsoft are in favor of a different storage mechanism and, given that together they currently have about 80% of the browser market (see chart below), that pretty much killed Web Database Storage.



In November, 2010, the W3C added this disclaimer to the Web SQL Database specification:



Status of This Document



Indexed Database API

The storage mechanism preferred by Mozilla and Microsoft is called Indexed Database API and is also a [W3C specification](#). It takes an object-approach rather than a relational database-approach to storage and querying. Google has also voiced support for it and has begun working on adding it to Chrome. Currently, only Firefox 4 beta has implemented Indexed Database API, so it will be some time before it's usable in applications.