Work offline with HTML5 web storage

Use web storage to add offline capabilities

Kris Hadlock

October 02, 2012

HTML5, the new HTML standard, supports a wide array of new functions and layout techniques. It fully supports multimedia, CSS3, and drawing capabilities with canvas and Scalable Vector Graphics (SVG). HTML5 offers new semantic elements, and also provides a legitimate way to create HTML web applications using application cache, JavaScript workers, a new version of XMLHttpRequest, and something called web storage. This article discusses the power of web storage and why it’s a better storage method than cookies. Learn about the basic concepts, browser support, and the HTML5 web storage objects.

Overview

Cookies have been around since the beginning of JavaScript, so storing data on the web isn’t a new concept. However, web storage is a much more powerful version of data storage that offers more security, speed, and ease of use. You can also store large amounts of data in web storage. The exact amount is based on the web browser, but it’s often between 5 and 10MB, which is a lot of storage for an HTML application. Another perk is that this data is not loaded with every server request. The only limitation is that you cannot share web storage between browsers; if you store data in Safari, that data is not accessible in Mozilla Firefox.

There are two types of web storage objects built in to HTML5:

- The sessionStorage object stores data for a single session. If the user closes the page or browser, the data is destroyed.
- The localStorage object stores data with no expiration date. The data remains stored when the web page or browser is closed, depending on the storage amount set for the user's browser.

Frequently used abbreviations

- API: Application Programming Interface
- CSS: Cascading Style Sheets
- HTML: HyperText Markup Language
- JSON: JavaScript Serialized Object Notation

Both storage objects have the same methods and properties. For consistency, this article uses the localStorage object throughout the examples.
In this article, learn about the power of web storage and why it is a better storage method than cookies. Explore the basic web storage concepts, HTML5 web storage methods, and browser support.

You can download the source code for the examples used in this article.

## Browser support

The web storage feature is supported by all new versions of the latest browsers, including Firefox, Google Chrome, Safari, Opera, and Microsoft® Windows® Internet Explorer® 8+. Unfortunately, Internet Explorer 7 and earlier do not support web storage. Table 1 shows the versions of each desktop browser that support HTML5 web storage.

### Table 1. Desktop browser support for HTML5 web storage

<table>
<thead>
<tr>
<th>Browser</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chrome</td>
<td>4+</td>
</tr>
<tr>
<td>Firefox</td>
<td>4+</td>
</tr>
<tr>
<td>Safari</td>
<td>4+</td>
</tr>
<tr>
<td>Opera</td>
<td>11+</td>
</tr>
<tr>
<td>Internet Explorer</td>
<td>8+</td>
</tr>
</tbody>
</table>

Mobile browsers, aside from Opera Mini, also provide support for HTML5 web storage. Table 2 shows the versions of each mobile browser that supports HTML5 web storage.

### Table 2. Mobile browser support for HTML5 web storage

<table>
<thead>
<tr>
<th>Browser</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>iOS</td>
<td>5+</td>
</tr>
<tr>
<td>Android</td>
<td>3+</td>
</tr>
<tr>
<td>Opera Mini</td>
<td>NA</td>
</tr>
<tr>
<td>Opera Mobile</td>
<td>11+</td>
</tr>
</tbody>
</table>

The browser support for HTML5 web storage is pretty impressive. However, older browsers require some sort of browser check for web storage support before you attempt to use it. Checking a browser for web storage support is easy. You can use a simple conditional statement to see if the HTML5 storage object is defined. If it is defined, it's safe to proceed with the web storage scripting. If it is undefined, you must use an alternative method, such as JavaScript cookies, if data storage is necessary. Listing 1 shows an example of a simple way to check the browser for the Storage object.

### Listing 1. Browser checking for web storage support

```javascript
if(typeof(Storage)!== "undefined") {
    // Web storage is supported
} else {
    // Web storage is NOT supported
}
```

Work offline with HTML5 web storage
If a browser doesn't support web storage, you can create a custom web storage object using either JavaScript cookies or an existing library such as AmplifyJS. AmplifyJS is a set of components designed to solve common web application problems, including web storage support in certain browsers, with a simplistic API. AmplifyJS addresses the issue with the `amplify.store` wrapper to handle persistent client-side storage, which supports Internet Explorer 5+, Firefox 2+, Safari 4+, Chrome, Opera 10.5+, iOS 2+, and Android 2+. The library also provides a consistent API to handle storage cross-browser; you don't have to write different code based on the browser in question. If the browser supports HTML5 web storage, AmplifyJS uses the latest storage techniques. If the browser doesn't support HTML5 web storage, AmplifyJS degrades to support it without that function. See Resources to learn more about AmplifyJS and the API for its storage wrapper.

**Getting started**

A few easy-to-use methods provide HTML5 web storage functions. These methods support setting a key/value pair, two options for retrieving a value based on a key, clearing all key/value pairs at once, and removing a specific key/value pair. Table 3 shows the available HTML5 web storage methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setItem(key, value)</code></td>
<td>Adds a key/value pair to the web storage object for later use. The value can be any data type: a string, number, array, and so on.</td>
</tr>
<tr>
<td><code>getItem(key)</code></td>
<td>Retrieves a value based on the key that was used to store it in the first place.</td>
</tr>
<tr>
<td><code>clear()</code></td>
<td>Clears all key/value pairs from the web storage object.</td>
</tr>
<tr>
<td><code>removeItem(key)</code></td>
<td>Removes a specific key/value pair from the web storage object based on a key.</td>
</tr>
<tr>
<td><code>key(n)</code></td>
<td>Retrieves the value for key[n].</td>
</tr>
</tbody>
</table>

When creating and adding key/value pairs to the web storage object, you can use any data type as the value in the pair (a string, number, array, object, and so on). To store an array or object, you must use the JSON object to convert the data to a string using the `JSON.stringify` method. When retrieving the data, you can parse it using `JSON.parse`, which returns your object or array in its original state. There are also two different ways to add a key/value pair to the web storage object. The first is to use the `setItem` method, as in Listing 2.

**Listing 2. Add a key/value pair to web storage object using `setItem`**

```javascript
localStorage.setItem('myKey', 'myValue');
```

The second option for adding a key/value pair to the web storage object is to set the key's value directly using the web storage object with dot syntax, as in Listing 3.

**Listing 3. Add a key/value pair to web storage object directly**

```javascript
localStorage.myKey = 'myValue';
```
Retrieving stored values is just as easy. Again, there are two options. The first is to use the `getItem` method, which takes a key as an argument and returns the corresponding value if one exists. Listing 4 shows an example.

**Listing 4. Retrieve a key/value pair from web storage object using `getItem`**

```javascript
localStorage.getItem('myKey');
```

The second option for retrieving a key/value pair from the web storage object is to access it directly using dot syntax, as in Listing 5. This example returns the `myValue` string value that was set in the previous examples.

**Listing 5. Retrieve a key/value pair from web storage object directly**

```javascript
localStorage.myKey;
```

There are also two ways to remove stored data. You can remove all items at once or you can remove individual items one at a time. To remove all items from web storage at once, use the `clear` method, as shown in Listing 6.

**Listing 6. Remove all key/value pairs from web storage object**

```javascript
localStorage.clear();
```

To remove a single key/value pair from a web storage object, you need to use the `removeItem` method. Listing 7 shows an example of the `removeItem` method that receives a key as an argument to identify what key/value pair to remove from the storage object.

**Listing 7. Remove a single key/value pair from web storage object**

```javascript
localStorage.removeItem('myKey');
```

Listing 8 shows an example of how to use the JSON object to store an array as a string using `JSON.stringify`. You handle objects the same way.

**Listing 8. Store an array as a string in HTML5 web storage**

```javascript
var myArray = new Array('First Name', 'Last Name', 'Email Address');
localStorage.formData = JSON.stringify(myArray);
```

To retrieve the string version of the array from web storage and convert it back to a usable array in JavaScript, simply use the `JSON.parse` method, as shown in Listing 9.

**Listing 9. Retrieve string version of an array from HTML5 web storage and convert it to a usable JavaScript array**

```javascript
var myArray = JSON.parse(localStorage.formData);
```
Internet Explorer 8+, Opera 10.5+, Firefox 3.5+, Safari 4+, and Chrome all include a native JSON object to support the code in the previous examples. If you use an earlier browser, you can download the json2.js file (see Resources).

As shown thus far, web storage is easy to use. However, before jumping in you should be aware that security can be a concern on shared machines. Web storage is not any more secure than cookies. Don't store sensitive information, such as passwords or credit card numbers, on the client side.

Web storage in action

Now that the basics are covered, it's time to put HTML5 web storage to use. Assume that, on your website, you have a web form for which you want to provide offline support. It would be great if a user could submit the form and have it sync with the server when the website is back online. This is possible with HTML5.

Create a simple web form that includes a first and last name, email address, and submit button, as shown in Listing 10.

Listing 10. Simple web form to store data using HTML5 web storage

```html
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>HTML5 Web Storage</title>
<style type="text/css">
label{
    display: block;
}
input{
    margin-bottom: 10px;
}
</style>
</head>
<body>
<form action="post.php" method="post" id="web-storage-form">
    <label for="first-name">First name:</label>
    <input type="text" name="first-name" id="first-name">
    <label for="last-name">Last name:</label>
    <input type="text" name="first-name" id="first-name">
    <label for="email-address">Email Address:</label>
    <input type="text" name="email-address" id="email-address">
    <input type="submit" value="Submit">
</form>
</body>
</html>
```

The form includes an ID to retrieve the form post and values using JavaScript. It also has CSS to create a basic layout with the form elements. The `display:block` property on labels and inputs
places each element on a new line. The margin-bottom property creates space between the items so the page doesn't look cluttered.

When a user submits the form, the code first prevents the default post from occurring by retrieving the web-storage-form ID and catching the post using jQuery. When the form posts, you can gather the form values and the URL of the form action to store in variables. You have to serialize the web form values when sending the values as an Asynchronous JavaScript + XML (Ajax) post or storing them in web storage. Before the form is submitted, use the navigator.onLine property to see if the user is currently online.

If the user is online, use the jQuery.post function, which is a shorthand Ajax function, to send and receive the data from the server. This function accepts four arguments: the url to send the data to, the data you are sending (the serialized form values), a callback function that is fired upon a successful request, and a dataType. In this example, the dataType is not included so it uses the default.

If the user is not online, you get fancy with web storage. First, it's important to see if the browser supports web storage by using the conditional statement you created in Listing 1. If the browser does support web storage, you can save the form values directly to the localStorage object using a custom key. The example uses the formValues custom key. Now that the saving of the localStorage values is in place, check to see if they exist when the user gets back online by adding an if statement to check if localStorage.formValues has a value. If it does, you know that this form was previously submitted to localStorage and can safely send the data to the server using the jQuery.post method that you set up earlier. After the values are submitted, you should remove them from web storage so they're not accidentally submitted a second time. Listing 11 shows the code in action, from the form post using Ajax to the localStorage.

Listing 11. Storing form data in localStorage while offline and submitting it to the server when online

```javascript
<script type="text/javascript">
$(document).ready(function() {
    // Check for web storage values from a previous offline session
    if(localStorage.formValues) {
        console.log("localStorage.formValues: "+ localStorage.formValues);
        postForm($("#web-storage-form").attr('action'), localStorage.formValues);
        localStorage.removeItem("formValues");
    }

    $('#web-storage-form').submit(function(event) {
        // Prevent the form from posting
        event.preventDefault();

        // Gather values
        var formValues = $(this).serialize();
        var url = $(this).attr('action');
        postForm(url, formValues);
    });
});

function postForm(url, formValues) {
```
To create a complete example, a post.php file acts as the end point of the form post to receive and respond to the form request. This file simply receives the form post and responds by printing the array of key/value pairs, as in Listing 12. When the response is received by the jQuery.post, the data from the response writes to the console.

Listing 12. PHP file that responds to form requests

```php
<?php print_r($_POST); ?>
```

Of course, you could make the example much more robust. For instance, you could include database storage on the server side and check the localStorage using an interval that constantly monitors whether the user's computer is back online to submit the form data.

Summary

HTML5 offers a powerful new set of functions that are quickly picking up support from the latest versions of the major web browsers. Web storage is one of the compelling features of HTML5. However, use it wisely. As with cookies, users can turn off web storage. Always have a fallback in place to support those who prefer not to allow this new function.
## Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work offline with web storage</td>
<td>html5-web-storage.zip</td>
<td>2KB</td>
</tr>
</tbody>
</table>

© Copyright IBM Corporation 2012  
**Trademarks**  
(www.ibm.com/developerworks/ibm/trademarks/)