Storing Data the Simple HTML5 Way (and a few tricks you might not have known)

Tuesday, August 23rd, 2011 by Remy Sharp.

Yes indeed people, it's your favourite HTML5 Doctor with JavaScript knowledge who lives in Brighton near a golf course! I'm also shooting for the longest title we've published so far – and I think I'm in the lead.

This post is about the Web Storage API. Technically it's been shifted out of the HTML5 specification and can now be found in it's very own dedicated spec. But if it counts at all – it *used* to be part of the Web Applications spec.
Web Storage is a very, **very** simple way to store data in the client – i.e. the browser. What's more, the support is fabulous: IE8 and upwards has support natively, and there's lots of good polyfills in the wild already.

This post however will just focus on the features of Web Storage and hopefully show you a trick or two you may not have known about.

What Web Storage Does

Using JavaScript, Web Storage makes it possible to easily store arbitrary values in the browser. Storage is different from cookies in that it's *not shared* with the server. It's also different from cookies in that it's dead simple to work with.
There are two versions of Web Storage: local and session. Local means that it's persistent, and session is simply that the data is lost when the session ends (i.e. when you close you browser window or tab). It's also worth noting that a session is tied to a single browser window or tab. The session *doesn't* leak out in to other open windows on that same domain.

Any data stored is tied to the document origin, in that it's tied to the specific protocol (http or https, etc), the host (html5doctor.com) and the port (usually port 80).

Get Storing

The API for `localStorage` and `sessionStorage` is exactly the same, and distills down to the following methods:

- .setItem(key, value);
- .getItem(key)
- .removeItem(key)
- .clear()
- .key(index)
- .length

You can probably guess what most of those methods do, maybe not the last two, but ignore all that syntax for a moment.

Due to the way that the storage API has been defined, the set/get/remove methods all are getters, setters and deleters. What that means to you is you can use localStorage as follows:

```
localStorage.name = 'Remy';
```

If you tried using the link above, this will save a property called “name” against localStorage. Completely closing your browser, and going back to http://jsconsole.com and test the name property again:

```
console.log(localStorage.name);
```

Still holds the value doesn’t it? Pretty easy eh? And no faffing around with cookies, for that, we can be thankful!

Equally deleting data is very easy:

```
delete localStorage.name;
console.log(localStorage.name);
```

You’ll see the value is undefined.

What’s happening under the hood is that when you set a property on `localStorage` (or `sessionStorage`) it’s calling the .setItem method. When you get a property, it’s calling .getItem and when you delete, it’s calling removeItem. That way you can treat the Web Storage interface as any other regular object in JavaScript, except you’ll know it’ll hang around after the page has unloaded.

Hey everyone! I’m storing some data!

That’s what your browser does when you store some data. It announces it via events.

When you set some data on localStorage or sessionStorage, an event, called “storage”, fires on all the other windows on the same origin.

To listen for events the following code is used:

```
function storageEvent(event) {
  event = event || window.event; // give IE8 some love
  alert('Yo people! Something just got stored!');
}

if (window.attachEvent) { // ::sigh:: IE8 support
   window.attachEvent('onstorage', storageEvent);
} else {
    window.addEventListener('storage', storageEvent, false);
}
```

Note that as IE8 has support, if you don’t want to use `onstorage` you’ll want to double up your event listeners with `attachEvent`.

Clearly a massive alert box isn’t useful, but what is useful is what you capture inside of the event object:

- key – the property name of the value stored
- newValue – the newly set value (duh!)
- oldValue – the previous value before being overwritten by .newValue
- url – the full url path of the origin of the storage event
- storageArea – the storage object, either localStorage or sessionStorage

With this information, you can do a lot with the storage event, like perhaps keep tabs that are open synchronised. Here’s a simple, contrived example of using localStorage to echo a value across different windows on html5demos.com.

Or for a real world example, Font Deck recently implemented this very feature, if you change the sample text in one window, all other open tabs (or windows) mirror that same sample text.

Gotchas to watch out for

The main gotcha with Web Storage is that although the specification *used to say*† that any object type can be stored, in fact all browsers currently coerce to strings. That means if you want to store a JavaScript object (or an array perhaps), you'll need to use JSON toencode and decode:

```
var doctors = [
  'rem',
  'rich_clark',
  'brucel',
  'jackosborne',
  'leads',
  'akamike',
  'boblet'];
localStorage.doctors = JSON.stringify(doctors);

// later that evening…
var html5docs = JSON.parse(localStorage.doctors);
alert('There be ' + html5docs.length + ' doctors in the house');
```

† Edited thanks to feedback from zcorpan

When working with storage events, a couple of things to also watch out for – this is on purpose of course, but still just about qualify for gotcha status:

1. The event only fires on the *other* windows – it won't fire on the window that did the storing.
2. The event won't fire if the data doesn't change, i.e. if you store .name = 'Remy' and set it to 'Remy' again it won't fire the storage event (obviously, since nothing was stored).

and finally…

If you want to use web storage in browsers like IE6 and IE7 (or rather than *want*: *need* to in your particular case) – then there's no shortage of polyfills for Web Storage. However be wary that you'll need to stick to the regular `setItem` syntax rather than being able to use the sexy setter/getter syntax – and I've yet to see a polyfill that supports the storage events.

Maybe that isn't a problem for you though – maybe the sheer simplicity and spiffingness of Web Storage is enough to just support IE8 and above with this enhanced client storage model.