# Chapter 15: Local data with web storage

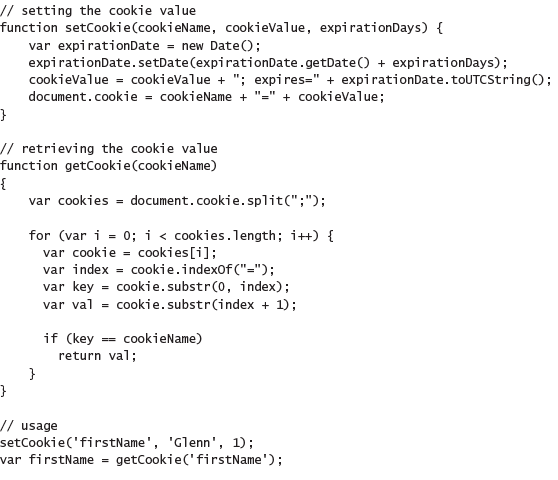
In this chapter we will be looking at two web storage mechanisms, localStorage and sessionStorage.

## Lesson 1: Introducing web storage

### Understanding cookies

Cookies provide a way for us to store **non-sensitive** information on the browser, so we do not have to make a trip to the server all the time.

The code below shows us how to Set and Get A Cookie:



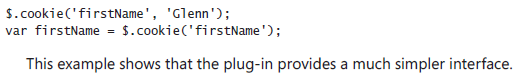
JSFiddle: https://jsfiddle.net/kmk6z95z/

**NOTE**: you must upload your file to a web server to set cookies on Google chrome (http://stackoverflow.com/questions/8105135/cannot-set-cookies-in-javascript)

### Using the jQuery cookie plug-in

The plugin can be found here: *https://github.com/carhartl/jquery-cookie*.

E.g.



### Working with cookie limitations

1. Capacity limitations: cookies are limited to about 4kb of data. Additionally, only 30 cookies can be created per site (actual number of cookies varies by browser; the average is between 30 and 50)

2. Overhead: **Every** cookie is sent with each HTTP request/response made, regardless of whether the values are needed.

### Alternatives to cookies prior to HTML5

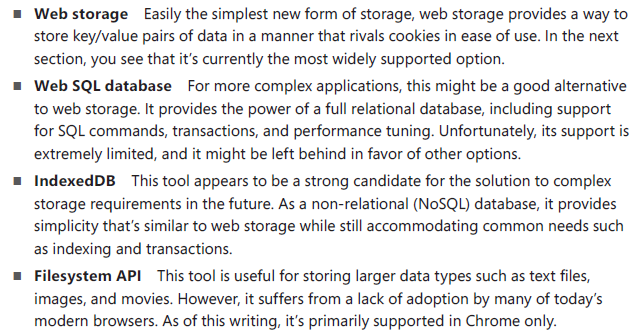
- Flash player  
- User Data (IE only)  
- Google Gears (support discontinued)  
- Java Applet

### Looking beyond plug-ins

The following are issues related to plug-in use:

* Plug-in & correct plug-in version required
* User extensions blocking plug-ins
* Corporate organization software restrictions
* Dependent on continued plug-in vendor support

## Understanding HTML5 storage

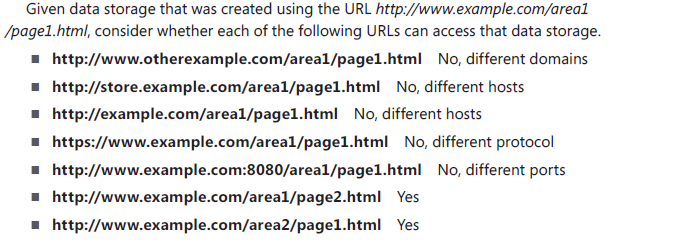


### Considering security

The major thing all four storage options have in common is that the data being stored is tied to the URL (or origin), to ensure that the data can't be accessed by other sites.

Therefore the same host, port, and protocol (http vs https) must be provided before a webpage can access data written by another page.

E.g.

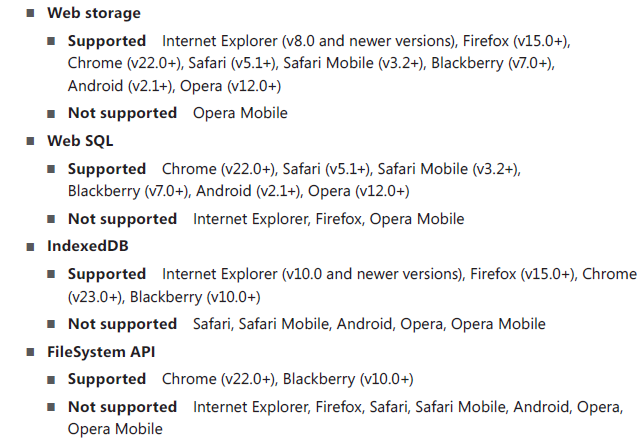


**Security Note**: It would be better practice to avoid using web storage entirely if your site will live within a shared domain in the future, because any sub-site within the domain would be able to access your data.

### Browser support

The most widely supported storage option is web storage.

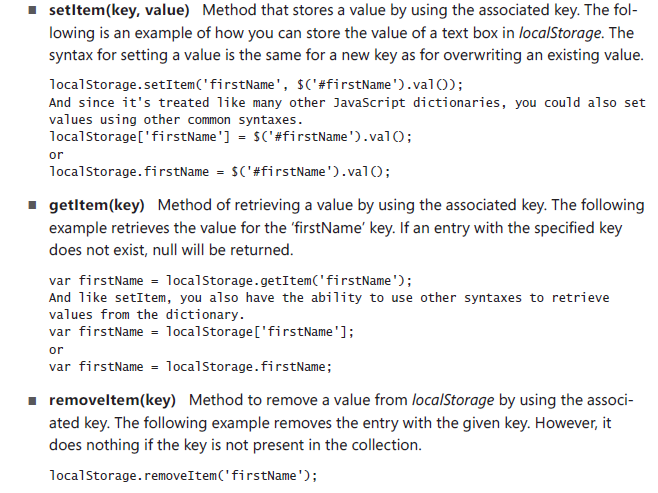
The list below shows the browser support for the 4 storage options:

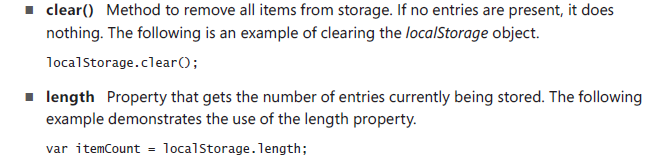
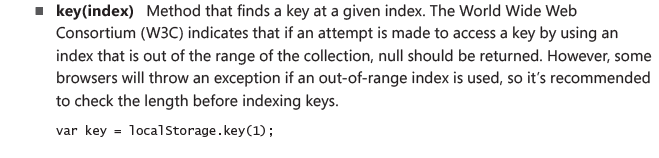


## Exploring localStorage

LocalStorage is a NoSQL, key/value storage option.

### Using the localStorage object reference

The following is a list of methods available on the localStorage object  


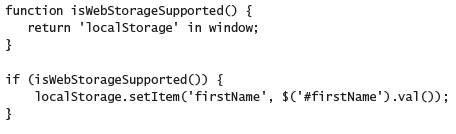
### Very high browser support

localStorage & sessionStorage are supported in modern browsers (both desktop and mobile browsers)

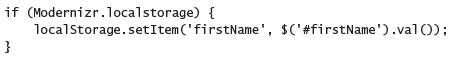
### Determining whether the user’s browser supports web storage

If localStorage or sessionStorage is not available, you could experience a null reference exception the first time an attempt is made to access localStorage or sessionStorage.

**Checking for localStorage using javascript**



**Checking for localStorage using Modernizr**



### Amount of data that can be kept in web storage

Modern browsers support a minimum of 5MB of data (more than the 4KB of data allowed by cookies).

**Note**: The 5 MB limit is currently recommended by the W3C, but it’s ultimately up to the browser vendors to determine how much they will allow. Currently, Internet Explorer supports a 10 MB limit.

### Reaching the storage limit

A *QuotaExceededError* exception is thrown when the storage limit is reached.

e.g. using a try/catch block to keep your app from failingwhen the storage limit is exceeded

try {

localStorage.setItem('firstName', $('#firstName').val());

}catch(e) {

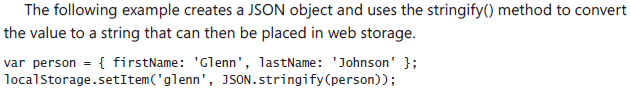
//degrade gracefully

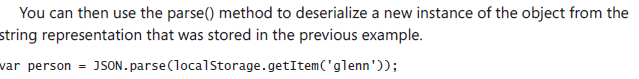
}

### Storing complex objects

Currently, only string values can be stored in web storage.

To store complex objects, we need to use the JSON utility methods.

**Storing a complex object in localStorage (JSON.stringify)**  


**Retrieving a complex object stored in the localStorage (JSON.parse)**  


**Note:** While cookies get sent in every HTTP request/response message, the values in the localStorage are never automatically sent to the server. You can send them yourself by including their values in an AJAX call or by using JavaScript to copy the values into posted form elements

## Using short-term persistence with sessionStorage

In the previous lesson, we learned that localStorage retains data across multiple sessions, and that to purge stored information, you must use the removeItem() or clear() method.

The sessionStorage object has all of the same methods and properties as the localStorage with one distinction: **sessionStorage retains data for a single session only**. **After the user closes the browser window, records are automatically cleared**.

Additionally, while the localStorage can be accessed from other tabs and windows (as long as the host, port, and protocol are the same), the sessionStorage data cannot be passed from one tab to another (however the data can be shared among any *<iframe>* elements that exist on the page

## Anticipating potential performance pitfalls

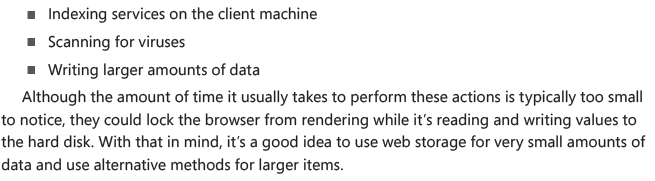
Below are some pitfalls of using localStorage & sessionStorage.

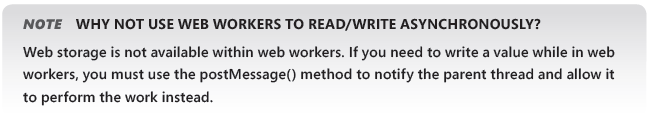
### Synchronous reading and writing to the hard drive

The localStorage & sessionStorage operate synchronously, which can block the page from rendering while read/writes occur.

The synchronous read/writes are even more costly because they are committed directly to the client's hard drive.

While by itself synchronous read/writes might not cause issues, the following activities can make these interactions annoyingly slow for the user:





### Anticipating slow search capabilities

Because web storage does not have indexing features, searching large data sets (which usually involves iterating over each item in the list), can be time consuming.

### No transaction support

Although difficulties are unlikely to occur in the majority of applications, applications using web storage can run into problems if a user is modifying the same value in localStorage within multiple open tabs. The result would be a race condition in which the second tab immediately overwrites the value inserted by the first tab.