DIG 4104C: WEB DESIGN WORKSHOP

Lecture 04: Extending web apps: File Editing & Management, Local storage

Summer 2015

D. Novatnak

Outline

Objectives

- Students will be exposed to local storage methods in HTML5 to extend web based applications
- Students will be exposed to file manipulation methods in HTML5 to extend web based applications
- Students will receive instruction on the language and commands necessary to store data and access files locally within the client's browser
- View and analyze code examples

At the end of this lecture, you should be able to….

- Store data on the client's browser for later retrieval with the Web Storage API
- Access data on local file systems from the client's browser for later retrieval with the File API
- Access data on the client's browser for processing through the Web Storage and File APIs

WEB STORAGE API

STORING OFFLINE FOR FUN AND PROFIT

Summary

- DOM Storage is the name given to the set of storage-related features first introduced in the Web Applications 1.0 specification, and now split off into its own W3C Web Storage specification.
- DOM Storage is designed to provide a larger, more secure, and easier-to-use alternative to storing information in cookies.
- First introduced with Firefox 2 and Safari 4.
- Implemented on desktop and mobile

Browser compatibility

- Desktop

Feature	Chrome	Mozilla (Gecko)	Internet Explorer	Opera	Safari (Webkit)
localStorage	4	3.5	8	10.5	4
sessionStorage	5	2	8	10.5	4

- Mobile

Feature	Android	Firefox mobile (Gecko)	Internet Explorer (Phone)	Opera Mobile	Safari Mobile
Basic Support	2.1	??	8	11	iOS 3.2

Web Storage concepts & usage

- The Web Storage API provides mechanisms by which browsers can securely store key/value pairs, in a much more intuitive fashion than using cookies.

- The two mechanisms within Web Storage are as follows:
  - sessionStorage
    - maintains a separate storage area for each given origin that's available for the duration of the page session (as long as the browser is open, including page reloads and restores)
  - localStorage
    - does the same thing, but persists even when the browser is closed and reopened.

Web Storage interfaces

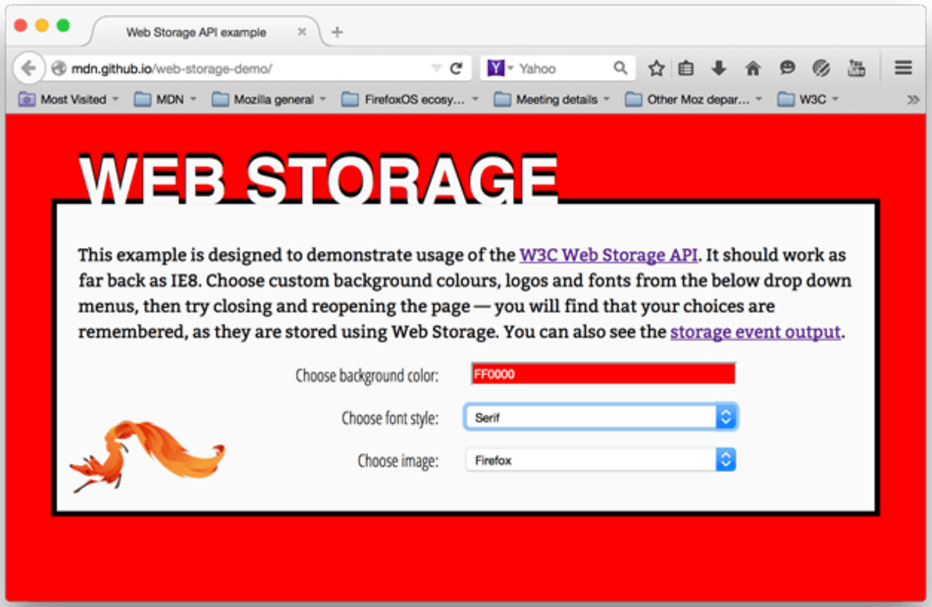
- Storage
  - Allows you to set, retrieve and remove data for a specific domain and storage type (session or local.)
- Window
  - The Web Storage API extends the Window object with two new properties
  - Window.sessionStorage and Window.localStorage which provide access to the current domain's session and local Storage objects respectively.
- StorageEvent
  - The storage event is fired on a Document's Window object when a storage area changes.

Window.sessionStorage & Window.localStorage properties

- In supporting browsers the Window object implements the WindowLocalStorage and WindowSessionStorage objects, which the localStorage and sessionStorage properties hang off
- Invoking one of these will create an instance of the Storage object, through which data items can be set, retrieved and removed.
- A different Storage object is used for the sessionStorage and localStorage, they function and are controlled separately.

Basic example from MDN

- To illustrate some typical web storage usage, MDN created a basic demo imaginatively called Web Storage Demo. (http://mdn.github.io/web-storage-demo/)
- The landing page provides controls that can be used to customize the color, font, and decorative image.



This example is designed to demonstrate usage of the W3C Web Storage API. It should work as far back as IE8. Choose custom background colours, logos and fonts from the below drop down menus, then try closing and reopening the page — you will find that your choices are remembered, as they are stored using Web Storage. You can also see the storage event output.

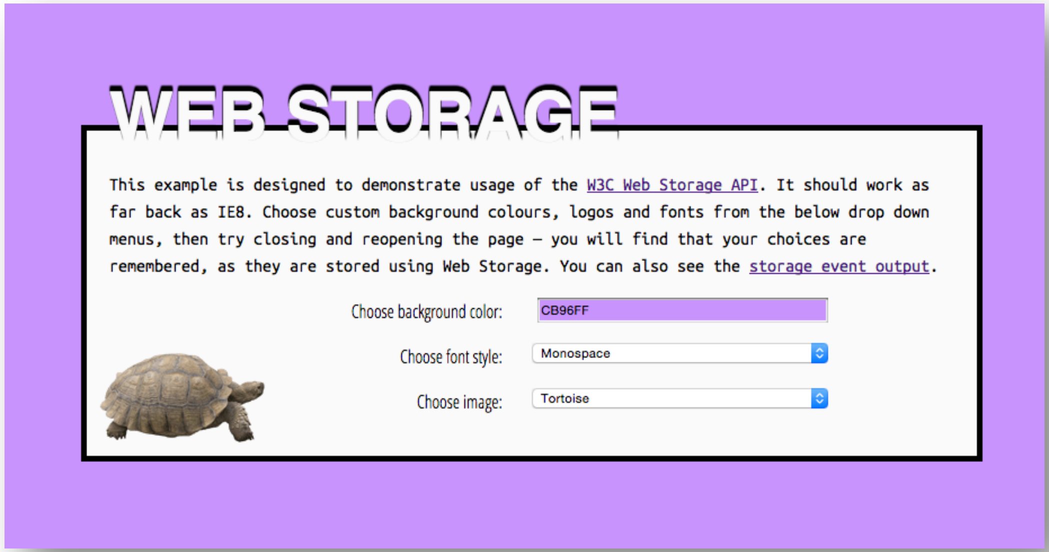
Choose background color: FF0000

Choose font style: Serif

Choose image: Firefox

Basic example from MDN

- When you choose different options, the page is instantly updated; in addition, your choices are stored in localStorage, so that when you leave the page and load it again later on, your choices are remembered.



This example is designed to demonstrate usage of the W3C Web Storage API. It should work as far back as IE8. Choose custom background colours, logos and fonts from the below drop down menus, then try closing and reopening the page – you will find that your choices are remembered, as they are stored using Web Storage. You can also see the storage event output.

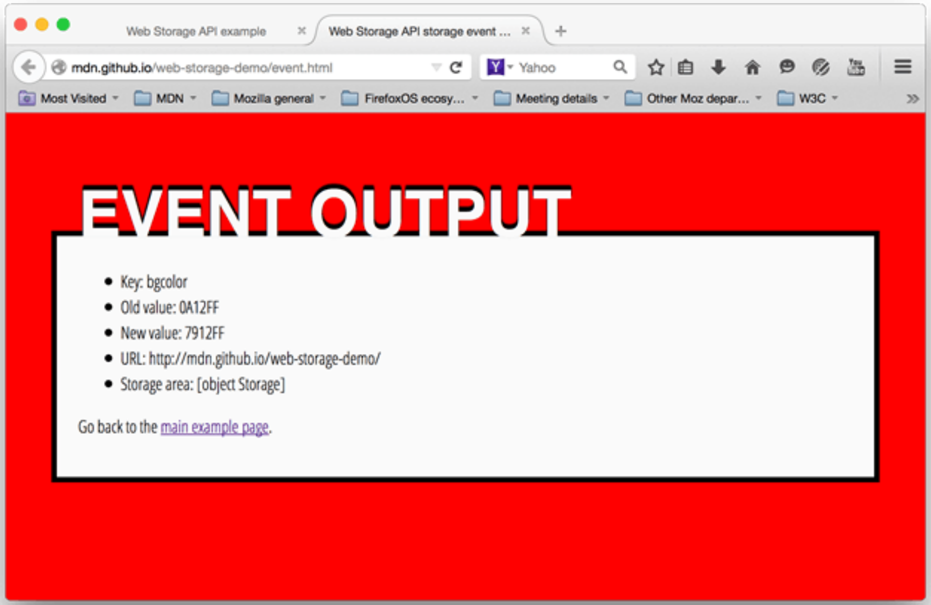
Choose background color: CB96FF

Choose font style: Monospace

Choose image: Tortoise

Basic example from MDN

- They also provided an event output page (http://mdn.github.io/web-storage-demo/event.html)
- Load this page in another tab, then make changes to the choices in the landing page, you'll see the updated storage information outputted as the StorageEvent is fired.



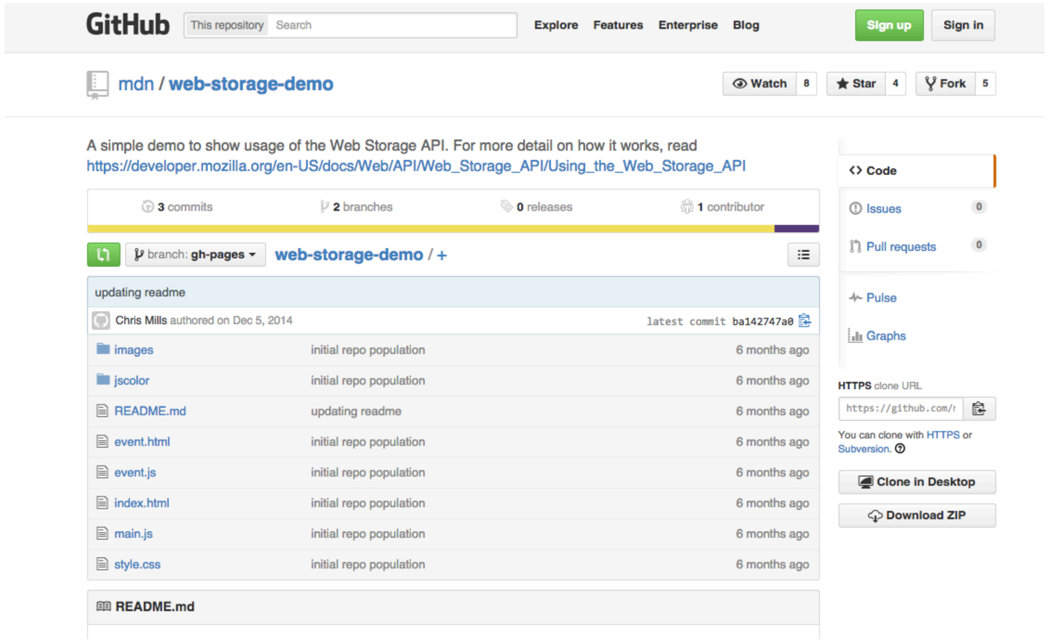
EVENT OUTPUT

- Key: bgcolor
- Old value: 0A12FF
- New value: 7912FF
- URL: http://mdn.github.io/web-storage-demo/
- Storage area: [object Storage]

Go back to the main example page.

Basic example from MDN

- As well as viewing the example pages live using the links, you can also check out the source code on GitHub (https://github.com/mdn/web-storage-demo)



mdn / web-storage-demo

A simple demo to show usage of the Web Storage API. For more detail on how it works, read https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API/Using_the_Web_Storage_API

3 commits · 2 branches · 0 releases · 1 contributor

branch: gh-pages · web-storage-demo / +

updating readme

Chris Mills authored on Dec 5, 2014 · latest commit ba142747a0

images	initial repo population	6 months ago
jscolor	initial repo population	6 months ago
README.md	updating readme	6 months ago
event.html	initial repo population	6 months ago
event.js	initial repo population	6 months ago
index.html	initial repo population	6 months ago
main.js	initial repo population	6 months ago
style.css	initial repo population	6 months ago

README.md

HTTPS clone URL

https://github.com/r

You can clone with HTTPS or Subversion.

Setting values from storage

- Storage.setItem() is used both to create new data items, and (if the data item already exists) update existing values.
- This takes two arguments; the key of the data item to create/modify, and the value to store in it.
- From the MDN example:

```
function populateStorage() {
  localStorage.setItem('bgcolor', document.getElementById('bgcolor').value);
  localStorage.setItem('font', document.getElementById('font').value);
  localStorage.setItem('image', document.getElementById('image').value);

  setStyles();
}
```

Getting values from storage

- Values can be retrieved from storage using Storage.getItem().
- This takes the key of the data item as an argument, and returns the data value.

- From the MDN example:

```
function setStyles() {
  var currentColor = localStorage.getItem('bgcolor');
  var currentFont = localStorage.getItem('font');
  var currentImage = localStorage.getItem('image');

  document.getElementById('bgcolor').value = currentColor;
  document.getElementById('font').value = currentFont;
  document.getElementById('image').value = currentImage;

  htmlElem.style.backgroundColor = '#' + currentColor;
  pElem.style.fontFamily = currentFont;
  imgElem.setAttribute('src', currentImage);
}
```

Testing if storage has been populated

- Testing whether your storage has been populated

- From the MDN example in main.js, there is a test whether the storage object has already been populated (i.e., the page was previously accessed):

```
if(!localStorage.getItem('bgcolor')) {
  populateStorage();
} else {
  setStyles();
}
```

Responding to storage changes with the StorageEvent

- The StorageEvent is fired whenever a change is made to the Storage object.
  - This won't work on the same page that is making the changes — it is really a way for other pages on the domain using the storage to sync any changes that are made.
- Pages on other domains can't access the same storage objects.

- From the example’s events page (events.js):

```
window.addEventListener('storage', function(e) {
  document.querySelector('.my-key').textContent = e.key;
  document.querySelector('.my-old').textContent = e.oldValue;
  document.querySelector('.my-new').textContent = e.newValue;
  document.querySelector('.my-url').textContent = e.url;
  document.querySelector('.my-storage').textContent = e.storageArea;
});
```

Deleting data records

- Web Storage also provides a couple of simple methods to remove data.

  - Storage.removeItem()
    - takes a single argument the key of the data item you want to remove
    - removes it from the storage object for that domain

  - Storage.clear()
    - takes no arguments
    - simply empties the entire storage object for that domain

FILE API

Creating, writing & editing files in the client browser

Introduction to the File API

- Web applications should have the ability to manipulate as wide as possible a range of user input, including files that a user may wish to upload to a remote server or manipulate inside a rich web application.

- The specification defines the basic representations for files, lists of files, errors raised by access to files, and programmatic ways to read files.

- Additionally, the specification also defines an interface that represents "raw data" which can be asynchronously processed on the main thread of conforming user agents.

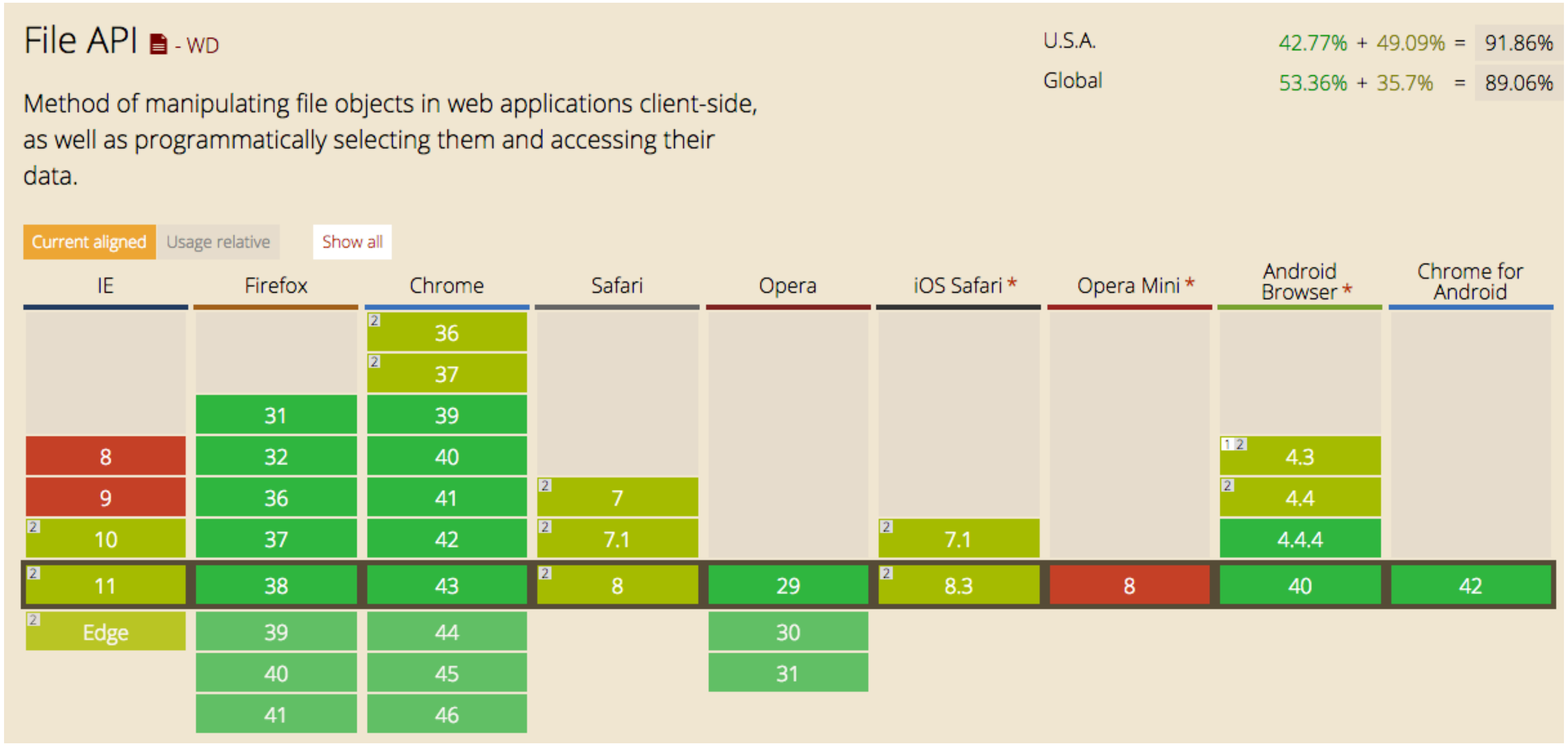
The File API

- The interfaces and API defined in the specification can be used with other interfaces and APIs exposed to the web platform.
- Using the File API, added to the DOM in HTML5, it's possible for web content to ask the user to select local files and then read the contents of those files.
- This selection can be done by either using an HTML <input> element or by drag and drop.
- This specification includes:
  - A FileList interface
  - A Blob interface
  - A File interface
  - A FileReader interface
  - A URL scheme

Files and Blobs

- The File interface represents file data typically obtained from the underlying (OS) file system, and the Blob interface represents immutable raw data.
- A FileReader object provides asynchronous read methods to access that file's data through event handler attributes and the firing of events.
- The use of events and event handlers allows separate code blocks the ability to monitor the progress of the read and error conditions that may arise during reading of a file.

Browser Compatibility



File API - WD

Method of manipulating file objects in web applications client-side, as well as programmatically selecting them and accessing their data.

U.S.A.	42.77%	+ 49.09%	=	91.86%
Global	53.36%	+ 35.7%	=	89.06%

Current aligned   Usage relative   Show all

IE	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
		[2] 36						
		[2] 37						
	31	39						
8	32	40					[1] [2] 4.3	
9	36	41	[2] 7				[2] 4.4	
[2] 10	37	42	[2] 7.1		[2] 7.1		4.4.4	
[2] 11	38	43	[2] 8	29	[2] 8.3	8	40	42
[2] Edge	39	44		30				
	40	45		31				
	41	46						

http://caniuse.com/#search=file

Accessing selected file(s)

- The File API makes it possible to access a FileList containing File objects representing the files selected by the user.
- If the user selects just one file, it is then only necessary to consider the first file of the list.

```
// The HTML:
<input type="file" id="input">

// Accessing one selected file using a classical DOM selector:
var selectedFile = document.getElementById('input').files[0];

//Accessing one selected file using a jQuery selector:
var selectedFile = $('#input').get(0).files[0];
var selectedFile = $('#input')[0].files[0];
```

Accessing selected file(s) on a change event

- It is also possible (but not mandatory) to access the FileList through the change event:

```
<input type="file" id="input" onchange="handleFiles(this.files)">

//If you want to let the user select multiple files, simply use the multiple attribute
on the input element:

<input type="file" id="input" multiple onchange="handleFiles(this.files)">
```

- When the user selects a file, the handleFiles() function gets called with a FileList object containing File objects representing the files selected by the user.

Sample JavaScript to access

```
function startRead() {
  // obtain input element through DOM

  var file = document.getElementById('file').files[0];
  if(file){
    getAsText(file);
  }
}

function getAsText(readFile) {

  var reader = new FileReader();

  // Read file into memory as UTF-16
  reader.readAsText(readFile, "UTF-16");

  // Handle progress, success, and errors
  reader.onprogress = updateProgress;
  reader.onload = loaded;
  reader.onerror = errorHandler;
}
```

```
function updateProgress(evt) {
  if (evt.lengthComputable) {
    // evt.loaded and evt.total are ProgressEvent properties
    var loaded = (evt.loaded / evt.total);
    if (loaded < 1) {
      // Increase the prog bar length
      // style.width = (loaded * 200) + "px";
    }
  }
}

function loaded(evt) {
  // Obtain the read file data
  var fileString = evt.target.result;
  // Handle UTF-16 file dump
  if(utils.regexp.isChinese(fileString)) {
    //Chinese Characters + Name validation
  }
  else {
    // run other charset test
  }
  // xhr.send(fileString)
}

function errorHandler(evt) {
  if(evt.target.error.name == "NotReadableError") {
    // The file could not be read
  }
}
```

Dynamically adding a change listener

- If your input field was created using a JavaScript library such as jQuery, you'll need to use element.addEventListener() to add the change event listener, like this:

```
var inputElement = document.getElementById("input");
inputElement.addEventListener("change", handleFiles, false);
function handleFiles() {
  var fileList = this.files; /* now you can work with the file list */
}
```

Using object URLs to display images

- Example:
  - https://developer.mozilla.org/samples/domref/file-click-demo.html
- The input:

```
<input type="file" id="fileElem" multiple accept="image/*" style="display:none"
onchange="handleFiles(this.files)">
<a href="#" id="fileSelect">Select some files</a>
<div id="fileList">
  <p>No files selected!</p>
</div>
```

The handleFiles() method

```
window.URL = window.URL || window.webkitURL;

var fileSelect = document.getElementById("fileSelect"),
    fileElem = document.getElementById("fileElem"),
    fileList = document.getElementById("fileList");

fileSelect.addEventListener("click", function (e) {
  if (fileElem) {
    fileElem.click();
  }
  e.preventDefault(); // prevent navigation to "#"
}, false);

function handleFiles(files) {
  if (!files.length) {
    fileList.innerHTML = "<p>No files selected!</p>";
  } else {
    fileList.innerHTML = "";
    var list = document.createElement("ul");
    fileList.appendChild(list);
    for (var i = 0; i < files.length; i++) {
      var li = document.createElement("li");
      list.appendChild(li);

      var img = document.createElement("img");
      img.src = window.URL.createObjectURL(files[i]);
      img.height = 60;
      img.onload = function() {
        window.URL.revokeObjectURL(this.src);
      }
      li.appendChild(img);
      var info = document.createElement("span");
      info.innerHTML = files[i].name + ": " + files[i].size + " bytes";
      li.appendChild(info);
    }
  }
}
```

Handler Explanaiton

- Begin by fetching the URL of the <div> with the ID fileList. This is the block into which we'll insert our file list, including thumbnails.
- If the FileList object passed to handleFiles() is null, we simply set the inner HTML of the block to display "No files selected!". Otherwise, we start building our file list, as follows:
- A new unordered list (<ul>) element is created.
  - The new list element is inserted into the <div> block by calling its element.appendChild() method.
  - For each File in the FileList represented by files:
  - Create a new list item (<li>) element and insert it into the list.
  - Create a new image (<img>) element.
  - Set the image's source to a new object URL representing the file, using window.URL.createObjectURL() to create the blob URL.
  - Set the image's height to 60 pixels.
  - Set up the image's load event handler to release the object URL since it's no longer needed once the image has been loaded. This is done by calling the window.URL.revokeObjectURL() method and passing in the object URL string as specified by img.src.
  - Append the new list item to the list.

HTML5 IN ACTION

Rich editor example from the book

Let’s take a look at the book example

- Html5 in Action has a great example of the usage of all the features of the File API
- http://html5inaction.com/app/ch3/#list

Questions?

References

- Crowther, R., & Lennon, J. (n.d.). File editing and management: rich formatting , file storage, drag and drop. In HTML5 in action (pp. 71-99).

- File API: Directories and System. (n.d.). Retrieved May 5, 2015, from http://www.w3.org/TR/FileAPI/

- Web Storage. (n.d.). Retrieved May 5, 2015, from http://dev.w3.org/html5/webstorage/