HTML5
Storage & Cache

@yorzi
andy@intridea.com

General Introduction

Background

World Wide Web Consortium (W3C)

+

Web Hypertext Application Technology Working Group (WHATWG)

History

- 1991 HTML
- 1994 HTML 2
- 1996 CSS 1 + JavaScript
- 1997 HTML 4
- 1998 CSS 2
- 2000 XHTML 1
- 2002 Tableless Web Design
- 2005 AJAX
- 2009 **HTML5 ~= HTML + CSS + JS**

Design Rules

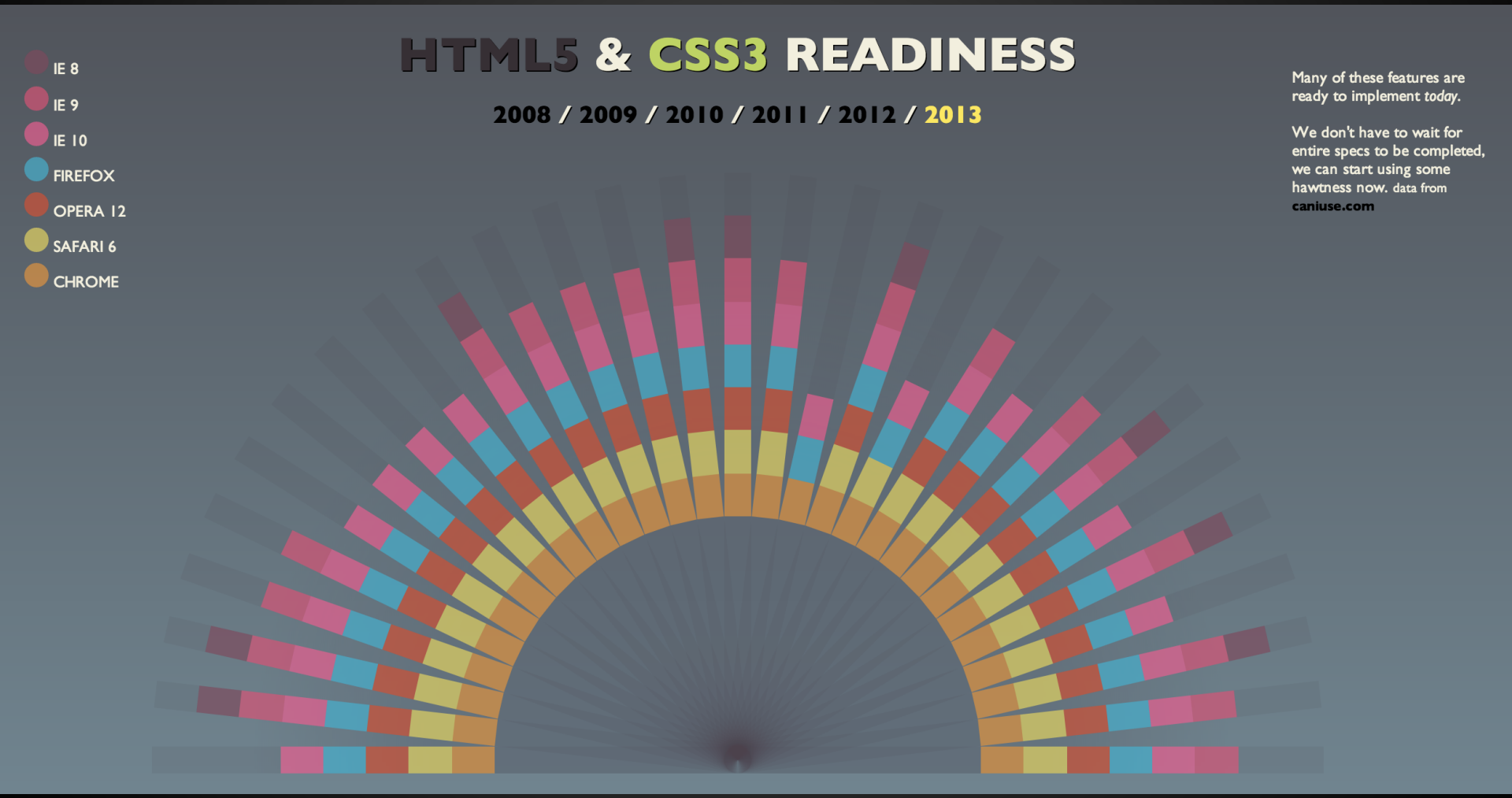
- Based on HTML, CSS, DOM, and JS
- Reduce the need for external plugins (Flash)
- Better error handling
- More markup to replace scripting
- HTML5 should be device independent
- The development process should be visible to the public

Minimum Sample

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Title of the document</title>
    </head>

    <body>
        Content of the document......
    </body>
</html>
```

Browser Compatiblity



Many of these features are ready to implement *today*.

We don't have to wait for entire specs to be completed, we can start using some hawtness now. data from **caniuse.com**

Features

Offline / Storage 

Realtime / Communication 

File / Hardware Access 

Semantics & Markup 

Graphics / Multimedia 

CSS3 

Nuts & Bolts 

Storage & Cache

Before HTML5

- Cookies
- Flash Storage
- Internet Explorer UserData
- Google Gears
- window.name (hack)

Storage In HTML5

- Web Storage APIs:
  localStorage / sessionStorage
- Web SQL Database
- IndexedDB
- Application Cache
- ...Cookies

window.localStorage

- Key/value hash table
- Persistent on page reloads
- Avoids cookies' limitations
- Scoped to an origin

```
http://example.com:80/
  \       \          \_ port
   \       \_ domain
    \_ scheme
```

localStorage API

```
localStorage.setItem('someKey', someValue);
localStorage.getItem('someKey'); // value

// Can also access as a property, but not recommended.
localStorage.someKey = someValue;

// Orther samples
localStorage.setItem('dateOfBirth', '1984-07-22');
localStorage.getItem('dateOfBirth'); // '1984-07-22'
localStorage.length // num of items stored
localStorage.key(0); // 'dateOfBirth'
localStorage.removeItem('dateOfBirth');
localStorage.clear();
```

window.sessionStorage

Same as localStorage but...

- Lasts as long as browser is open
- Opening page in new window or tab starts new session

Web Storage APIs Limitation

Web Storage APIs only store strings!

Solution:

```
localStorage.setItem('user', JSON.stringify({user: 'john', id: 10}));

var user = JSON.parse(localStorage.getItem('user'));
```

Web SQL Database

- Client-side SQL database
- Asynchronous & Synchronous* API
- Basically wrapper around SQLite

window.openDatabase();

```
var db = window.openDatabase(
  'MyDB',           // dbName
  '1.0',            // version
  'test database',  // description
  2 * 1024 * 1024,  // estimatedSize in bytes
  function(db) {}   // optional creationCallback
);
```

window.executeSql();

```
tx.executeSql(sqlStatement, opt_arguments, opt_callback, opt_errorCb);
```

Create:

```
tx.executeSql('CREATE TABLE IF NOT EXISTS ' +
    'todo(id INTEGER PRIMARY KEY ASC, task TEXT, added_on DATETIME)');
```

Insert:

```
var taskText = 'buy groceries';
tx.executeSql(
  'INSERT INTO todo(task, added_on) VALUES (?,?)', [taskText, new Date()],
  renderFunc, onError);
```

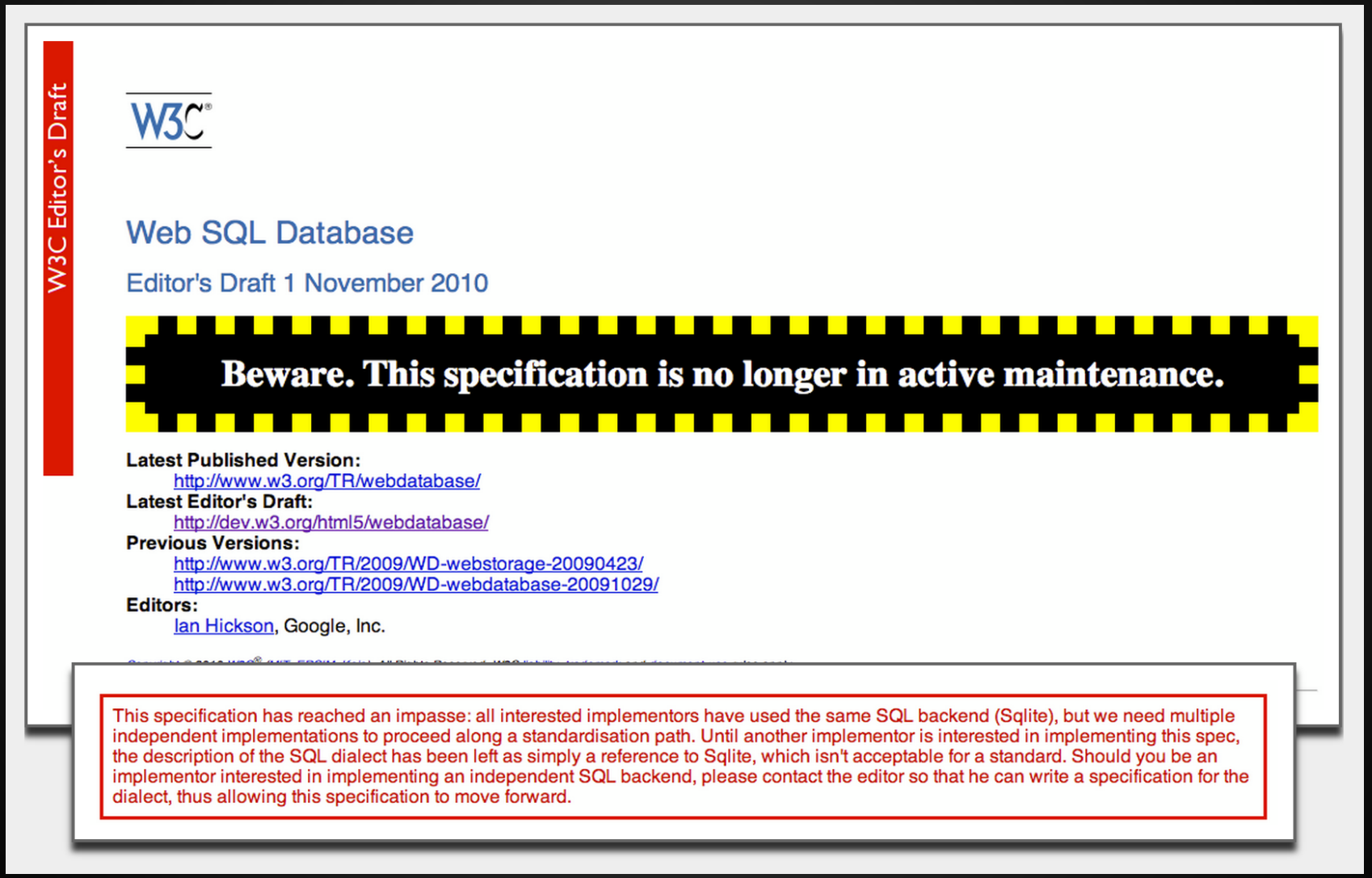
Delete:

```
tx.executeSql('DELETE FROM todo WHERE id=?', [id], renderFunc, onError);
```

Transaction

- `readTransaction`
  - read-only operations
  - ```
    db.readTransaction(function(tx) {
      // executeSql
    }, opt_errorCallback, opt_successCallback);
    ```
- `transaction`
  - read/write operations
  - Locks the entire database
  - ```
    db.transaction(function(tx) {
      // executeSql
    }, opt_errorCallback, opt_successCallback);
    ```

Deprecated


W3C Editor's Draft
W3C
Web SQL Database
Editor's Draft 1 November 2010
Beware. This specification is no longer in active maintenance.
Latest Published Version:
http://www.w3.org/TR/webdatabase/
Latest Editor's Draft:
http://dev.w3.org/html5/webdatabase/
Previous Versions:
http://www.w3.org/TR/2009/WD-webstorage-20090423/
http://www.w3.org/TR/2009/WD-webdatabase-20091029/
Editors:
Ian Hickson, Google, Inc.
This specification has reached an impasse: all interested implementors have used the same SQL backend (Sqlite), but we need multiple independent implementations to proceed along a standardisation path. Until another implementor is interested in implementing this spec, the description of the SQL dialect has been left as simply a reference to Sqlite, which isn't acceptable for a standard. Should you be an implementor interested in implementing an independent SQL backend, please contact the editor so that he can write a specification for the dialect, thus allowing this specification to move forward.

Indexed Database

- Object based data store
- In-order retrieval by index or locate by key
- Asynchronous & Synchronous API

Object Store

```
var db = window.indexedDB.open('IntrideaDB', 1);

if (db.version != '1') {
  // User's first visit, initialize database.
  db.createObjectStore('Friends', // name of new object store
                       'id',      // key path
                       true);     // auto increment?
  db.setVersion('1');
} else {
  // DB already initialized.
}
```

Put into Store

```
var store = db.openObjectStore('Friends');

var user = store.put({name: 'Eric',
                      gender: 'male',
                      likes: 'html5'});

console.log(user.id); // Expect 1
```

Retrieving by key (indexes)

```
db.createIndex(indexName, keyPath, unique?);

// db.createObjectStore("Friend", "id", true);
db.createIndex("FriendNames", "name", false);
var index = db.openIndex('FriendNames');
var id = index.get('Eric');
```

Querying (cursors)

```
db.createIndex(indexName, keyPath, unique?);

// db.createObjectStore("Friend", "id", true);
db.createIndex("FriendNames", "name", false);
var index = db.openIndex('FriendNames');
var id = index.get('Eric');
```

Web SQL DB VS. Indexed DB

Category	WebSQL	IndexedDB
Advantages	A real, relational database implementation on the client (SQLite).	* Allows fast indexing and searching of objects, so in a web application scenario, you can manage your data and read/write it fast. * A NoSQL database that let you work with your JavaScript objects and indexing them based on your application needs. * Works in asynchronous mode with moderately granular locking per transaction. This allows you to work inside the event-driven module of JavaScript.
Disadvantages	* The spec is deprecated. * Overhead of SQL language you need to master and transform your JavaScript objects into relational schema * Not object driven	Harder to understand if you are coming from the world of relational databases.
Location	Tables contain columns and rows	objectStore contains Javascript objects and keys
Query Mechanism	SQL	Cursor APIs , Key Range APIs , and Application Code
Transaction	Lock can happen on databases, tables, or rows on 'readwrite' transactions	Lock can happen on database 'versionchange' transaction, on an objectStore 'readonly' and 'readwrite' transactions.
Transaction Commits	Transaction creation is explicit. Default is to rollback unless we call commit.	Transaction creation is explicit. Default is to commit unless we call abort or there is an error that is not caught.

Application Cache (Why?)

* HTML, CSS, and JS stay fairly consistent
* Native browser caching is unreliable
* Caching resources creates speedier apps!
* Decent mobile support

Menifest File

```
<html manifest="example.appcache">... </html>
```

```
CACHE MANIFEST
# 2010-11-17-v0.0.1

# Explicitly cached entries
CACHE:
index.html
stylesheet.css
images/logo.png
http://img.example.com/logo2.png
scripts/main.js

# static.html will be served if the user is offline
FALLBACK:
/ /static.html

# Resources that require the user to be online.
NETWORK:
*
# login.php, http://api.twitter.com, etc.
```

Server with mime-type: `text/cache-manifest`

JS API

```
var cache = window.applicationCache;
```

```
cache.addEventListener('cached', handleCacheEvent, false);
cache.addEventListener('checking', handleCacheEvent, false);
cache.addEventListener('downloading', handleCacheEvent, false);
cache.addEventListener('error', handleCacheError, false);
cache.addEventListener('noupdate', handleCacheEvent, false);
cache.addEventListener('obsolete', handleCacheEvent, false);
cache.addEventListener('progress', handleCacheEvent, false);
cache.addEventListener('updateready', handleCacheEvent, false);

// When a new manifest is successfully downloaded, swap the new cache and reload p
cache.addEventListener('updateready', function(e) {
  if (cache.status == cache.UPDATEREADY) {
    cache.swapCache();
    if (confirm('A new version of this site is available. Load it?')) {
      window.location.reload();
    }
  }
}, false);
```

Detecting Support


```
if (Modernizr.localstorage) { ... }
if (Modernizr.sessionstorage) { ... }
if (Modernizr.websqldatabase) { ... }
if (Modernizr.indexeddb) { ... }
if (Modernizr.applicationcache) { ... }
```

Resources

http://www.html5rocks.com/en/resources

http://html5labs.interoperabilitybridges.com

http://html5-demos.appspot.com/static/html5storage/index.html

Thanks !