Support Offline Mobile Web Apps with HTML5 API's

Kim Dalsgaard

@kimdalsgaard

Application Cache

Local Storage

Web SQL Database

Indexed Database

Application Cache

Offline Web applications

From the specification

In order to enable users to continue interacting with Web applications and documents even when their network connection is unavailable —

authors can provide a manifest which lists the files that are needed for the Web application to work offline and which causes the user's browser to keep a copy of the files for use offline.

Jake Archibald

Application Cache is a Douchebag

Douchebag

An individual who has an over-inflated sense of self worth, compounded by a low level of intellegence, behaving ridiculously in front of colleagues with no sense of how moronic he appears.

The manifest Attribute

```
<!DOCTYPE html>
<html manifest='cache.manifest'>
    ...
</html>
```

Referencing the manifest file 'cache.manifest'

The referring file will go into the cache as a Master Entry

The Manifest File

```
CACHE MANIFEST
# Version 1

CACHE:

FALLBACK:

NETWORK:
```

Manifests must be served using the text/cache-manifest MIME type.

```
Content-type: text/cache-manifest
```

CACHE MANIFEST

```
CACHE MANIFEST
# Version 1
```

A manifest file should always start with the text CACHE MANIFEST as the very first line.

No blank lines before are allowed.

Lines starting with a # is a comment line.

CACHE:

```
CACHE:

/stylesheets/style.css
/javascripts/script.js
...
```

Resources that should be part of the cache.

The resources has to be loaded with the same protocol as the manifest file.

But do not have to come from det same origin.

FALLBACK:

```
FALLBACK:

/images/cars/ /images/cars/graycar.jpg
/images/cars/greencar.jpg /images/cars/bluecar.jpg
```

Fallbacks for resources not in the cache.

The fallback resource will go into the cache as a Fallback Entry.

The fallback resource has to be loaded from the same origin as the manifest file.

NETWORK:

```
NETWORK:
/stylesheets/
/javascripts/script.js
http://openexchangerates.org/api/
```

```
NETWORK:
*
```

An online whitelist. The URL's in this section is allowed to be accessed from the network.

The resources has to be loaded with the same protocol as the manifest file.

An asterisk means all resources with the same protocol.

Implicit CACHE: Section

```
CACHE MANIFEST

/stylesheets/style.css
```

is a shorthand for

```
CACHE MANIFEST

CACHE:

/stylesheets/style.css
```

The Browser Cache

The Browser Cache is working 'normally' behind the Application Cache.

Both when getting resources not in the application cache, and when refreshing the application cache.

The 'Double Refresh' Deal

When a page under application cache control is reloaded, the page is rendered from the application cache simultaneously with the manifest being checked for changes.

If the manifest file i changed, the new cache is downloaded, but the content is not shown before the next page reload.

The Manifest is Gone!?

If the request for cache manifest returns a 404 (Not Found) or a 410 (Gone), the cache is deleted.

Application Cache API

The applicationCache property on the window object is the main object representing the Application Cache to which this Master Entry belongs.

Properties

```
window.applicationCache.status
```

Returns the current status of the application cache, as given by the constants defined below.

```
const unsigned short UNCACHED = 0;
const unsigned short IDLE = 1;
const unsigned short CHECKING = 2;
const unsigned short DOWNLOADING = 3;
const unsigned short UPDATEREADY = 4;
const unsigned short OBSOLETE = 5;
```

Methods

```
window.applicationCache.update()
```

Invokes the application cache download process.

```
window.applicationCache.abort()
```

Cancels the application cache download process.

```
window.applicationCache.swapCache()
```

Switches to the most recent application cache, if there is a newer one. If there isn't, throws an InvalidStateError exception.

Events

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

```
cache.addEventListener('checking', function() {
  log.innerHTML += "Checking\n";
}, false);
```

Checking for an update, or attempting to download the manifest for the first time. This is always the first event in the sequence.

```
cache.addEventListener('noupdate', function() {
  log.innerHTML += "No Update\n";
}, false);
```

The manifest hadn't changed.

```
cache.addEventListener('downloading', function() {
  log.innerHTML += "Downloading\n";
}, false);
```

Found an update and is fetching it, or is downloading the resources listed by the manifest for the first time.

```
cache.addEventListener('progress', function() {
  log.innerHTML += "*";
}, false);
```

Downloading resources listed by the manifest.

```
cache.addEventListener('cached', function() {
  log.innerHTML += "\nCached\n";
}, false);
```

The resources listed in the manifest have been downloaded, and the application is now cached.

```
cache.addEventListener('updateready', function() {
  if (confirm("Update ready - reload?")) {
    location.reload();
  }
}, false);
```

The resources listed in the manifest have been reloaded, and the application is now ready to get swapped.

```
cache.addEventListener('obsolete', function() {
  log.innerHTML += "Obsolete\n";
}, false);
```

The manifest was found to have become a 404 or 410 page, so the application cache is being deleted.

```
cache.addEventListener('error', function() {
  alert("Error");
}, false);
```

A fatal error occurred while fetching the resources listed in the manifest.

The manifest changed while the update was being run.

Local Storage

An API for persistent data storage of key-value pair data in Web clients

From the specification

The second storage mechanism [Local Storage] is designed for storage that spans multiple windows, and lasts beyond the current session.

The localStorage Attribute

The localStorage attribute provides a Storage object for an given origin.

User agents must have a set of local storage areas, one for each origin.

The Local Storage Interface

The Local Storage object provides access to a list of key/value pairs, which are sometimes called items.

Keys are strings. Any string (including the empty string) is a valid key.

Values are similarly strings.

```
window.localStorage.getItem(key)
```

This method will return the current value associated with the given key. If the given key does not exist in the list associated with the object then this method will return null.

```
window.localStorage.setItem(key, value)
```

This method will first check if a key/value pair with the given key already exists in the list associated with the object.

If it does not, then a new key/value pair will be added to the list, with the given key and with its value set to value.

If the given key does exist in the list, then it will have its value updated to value.

```
window.localStorage.removeItem(key)
```

This method will cause the key/value pair with the given key to be removed from the list associated with the object, if it exists.

If no item with that key exists, the method must do nothing.

```
window.localStorage.clear()
```

This method will atomically cause the list associated with the object to be emptied of all key/value pairs, if there are any.

If there are none, then the method must do nothing.

```
window.localStorage.length
```

This attribute will return the number of key/value pairs currently present in the list associated with the object.

```
window.localStorage.key(n)
```

This method must return the name of the nth key in the list.

If n is greater than or equal to the number of key/value pairs in the object, then this method must return null.

Syntactic Sugar

```
window.localStorage.getItem(key)
```

Can be written as

```
window.localStorage[key]
```

And

```
window.localStorage.setItem(key, value)
```

Can be written as

```
window.localStorage[key] = value
```

Local Storage Events

```
window.addEventListener('storage', function(event) {
  event.key;
  event.oldValue;
  event.newValue;
  event.url;
  event.storageArea;
}, false);
```

Limitations

Only one key/value map

Synchronous API

Web SQL Database

An API for storing data in databases that can be queried using a variant of SQL

From the specification

This specification introduces a set of APIs to manipulate client-side databases using SQL.

The API is asynchronous, so authors are likely to find anonymous functions (lambdas) very useful in using this API.

Status

Beware. This specification is no longer in active maintenance and the Web Applications Working Group does not intend to maintain it further.

Opening a Database

```
var db = openDatabase('db', '1', 'Database', 2097152);
```

```
var db = openDatabase('db', '1', 'Database', 2097152, fu
nction(db) {
  // Initialize the database
});
```

```
var db = openDatabase('db', '', 'Database', 2097152);
```

Version Migration

```
var db = openDatabase('db', '', 'Database', 2097152);

if (db.version != 2) {
  db.changeVersion(db.version, 2, change, error, success
);
}

function change(transaction) {
  var version = db.version ? parseInt(db.version) : 0;
  if (version < 1) { /* Migrate to version 1 */ }
  if (version < 2) { /* Migrate to version 2 */ }
}
```

```
function error(e) {
  // Something went wrong
  console.log(e);
}

function success() {
  // Everything went right
  console.log("Upgraded to version " + db.version);
}
```

Changing the Database Schema

```
function change(tx) {
  if (db.version < 1) {
    tx.executeSql('CREATE TABLE foo (id unique, text)');
  }
  if (db.version < 2) {
    tx.executeSql('ALTER TABLE foo RENAME TO bar');
  }
}
```

Creating a Read-Write Transaction

```
db.transaction(function (tx) {
  // Do SQL stuff her
});
```

```
db.transaction(function (tx) {
  // Do SQL stuff her
}, error, success);
```

Creating a Read-Only Transaction

```
db.readTransaction(function (tx) {
  // Do SQL stuff her
});
```

```
db.readTransaction(function (tx) {
  // Do SQL stuff her
}, error, success);
```

Executing SQL

```
db.transaction(function(tx) {
  tx.executeSql('INSERT INTO bar (id, text) VALUES (?, ?
)', [1, "foo"]);
}, error);

function error(e) {
  console.log(e);
}
```

Getting a Result Set back

```
db.readTransaction(function(tx) {
  tx.executeSql('SELECT * FROM bar', [], success, error)
;
});

function success(tx, resultSet) {
  // Read resultSet
}
```

Reading a Result Set

```
function success(tx, rs) {
  var len = rs.rows.length, i;
  for (i = 0; i < len; i++) {
    var item = rs.rows.item(i);
    console.log(item.text);
  }
}
```

Indexed Database API

An Indexed Key-Value Store

From the specification

APIs for a database of records holding simple values and hierarchical objects. Each record consists of a key and some value. Moreover, the database maintains indexes over records it stores.

An application developer directly uses an API to locate records either by their key or by using an index. A query language can be layered on this API.

IndexedDB Request Pattern

```
var req = an_idb_object.some_operation();

req.onsuccess = function(e) {
  // Do something with this.result;
};

req.onerror = function(e) {
  // Do something with this.error;
};
```

Opening a Database

```
var db;
var req = window.indexedDB.open("db", 1);
req.onsuccess = function(e) {
  db = this.result;
};
req.onupgradeneeded = function(e) {
  // Upgrade the database
};
req.onerror = function(e) {
  console.log(e);
};
```

Upgrading a Database

```
req.onupgradeneeded = function(e) {
  var db = this.result;
  var version = e.oldVersion;
  if (version < 1) {
    db.createObjectStore("foo");
  }
  if (version < 2) {
    db.createObjectStore("bar");
  }
};
```

Creating a Object Store

Without a Key Path and without a Key Generator

```
var store = db.createObjectStore("foo");
```

With a Key Path but without a Key Generator

```
var store = db.createObjectStore("foo", {
  keyPath: "key"
});
```

Without a Key Path but with a Key Generator

```
var store = db.createObjectStore("foo", {
  autoIncrement: true
});
```

With a Key Path and with a Key Generator

```
var store = db.createObjectStore("foo", {
  keyPath: "_id",
  autoIncrement: true
});
```

Creating an Index

An Non-Unique Index

```
store.createIndex("name", "name");
```

An Unique Index

```
store.createIndex("email", "primaryEmail", { unique: tru
e });
```

A Multi Entry Index

```
store.createIndex("email", "emails", { multiEntry: true
});
```

Creating a Transaction

A Read-Only Transaction spanning 'foo' and 'bar'

```
var t = db.transaction(["foo", "bar"]);
```

A Read-Write Transaction spanning all Object Stores

```
var t = db.transaction([], 'readwrite');
```

A Read-Only Transaction spanning only 'baz'

```
var t = db.transaction("baz", 'readonly');
```

Transaction Callbacks

```
var transaction = db.transaction([], 'readwrite');
transaction.oncomplete = function(e) {
  // The transaction is completed
};
transaction.onerror = function(e) {
  // An error occurred during the transaction
};
transaction.onabort = function(e) {
  // The transaction was aborted
};
```

Error Events Bubbles

```
var transaction = db.transaction([], 'readwrite');

db.onerror = function(e) {
  // Will get called by transaction errors
};
```

Adding Data to a Object Store

```
var tx = db.transaction("foo", 'readwrite');
var store = tx.objectStore("foo");
var req = store.add(data);
req.onsuccess = function(e) {
  console.log(this.result);
};
```

Adding an Object with an Explicit Key

```
var req = store.add(data, key);
```

Note that the add() function requires that no object already be in the database with the same key. If you're trying to modify an existing entry, or you don't care if one exists already, use the put() function.

```
var req = store.put(data, key);
```

Getting an Object from an Object Store

```
var tx = db.transaction("foo");
var store = tx.objectStore("foo");
var req = store.get("123-4567-89");
req.onsuccess = function(e) {
  console.log(this.result);
};
```

A compact way of writing the same expression

```
db.transaction("foo")
  .objectStore("foo")
  .get("123-4567-89")
  .onsuccess = function(e) {
    console.log(this.result);
  };
```

Assuming that errors are handled at the database level.

Removing an Object from an Object Store

```
var tx = db.transaction("foo");
var store = tx.objectStore("foo");
var req = store.delete("123-4567-89");
req.onsuccess = function(e) {
  // It's gone!
};
```

Prevent Default

```
var req = store.add(data, key);
req.onsuccess = function(e) {
  console.log(this.result);
};
req.onerror = function(e) {
  e.preventDefault();
};
```

Opening a Cursor

```
var tx = db.transaction("foo", "readwrite");
var store = tx.objectStore("foo");
store.openCursor().onsuccess = function(e) {
  var cursor = this.result;
  if (cursor) {
    // Use cursor
    cursor.continue();
  }
};
```

Using a Cursor

```
function toMap(store, callback) {
  var map = {};
  store.openCursor().onsuccess = function(e) {
    var cursor = this.result;
    if (cursor) {
      map[cursor.key] = cursor.value;
      cursor.continue();
    } else {
      callback(map);
    }
  };
}
```

Using an Index

```
var tx = db.transaction("foo", "readonly");
var store = tx.objectStore("foo");
var index = store.index("odd");
index.get(1).onsuccess = function(e) {
  console.log(this.result);
};
```

If the 'odd' index isn't a unique index, then there could be more than one entry with the value of 1.

In that case you always get the one with the lowest key value.

Opening a Cursor on an Index

```
index.openCursor().onsuccess = function(e) {
  var cursor = this.result;
  if (cursor) {
    // key, value, primaryKey
    cursor.continue();
  }
}
```

Opening a Key Cursor on an Index

```
index.openKeyCursor().onsuccess = function(e) {
  var cursor = this.result;
  if (cursor) {
    // key, primaryKey
    cursor.continue();
  }
}
```

Using a Key Range on an Index

```
var keyRange = IDBKeyRange.only(1);
index.openCursor(keyRange).onsuccess = function(e) {
  var cursor = this.result;
  if (cursor) {
    // key == 1, value, primaryKey
    cursor.continue();
  }
}
```

```
var keyRange = IDBKeyRange.only(4);
```

key == 4

```
var keyRange = IDBKeyRange.lowerBound(4);
```

key >= 4

```
var keyRange = IDBKeyRange.lowerBound(4, true);
```

key > 4

```
var range = IDBKeyRange.upperBound(4);
```

key <= 4

```
var range = IDBKeyRange.upperBound(4, true);
```

key < 4

```
var range = IDBKeyRange.bound(4, 10);
```

key >= 4 and key <= 10

```
var range = IDBKeyRange.bound(4, 10, false, true);
```

key >= 4 and key < 10

Changing a Cursors Direction

```
var cursor = index.openCursor(null, 'next');
```

```
var cursor = index.openCursor(null, 'prev');
```

```
var cursor = index.openCursor(null, 'nextunique');
```

```
var cursor = index.openCursor(null, 'prevunique');
```

Database Blocking

```
var req = window.indexedDB.open("db", 2);
req.onblocked = function(e) {
  alert("Please close all other tabs!");
};
```

Vendor Prefix

```
window.indexedDB = window.indexedDB ||
                   window.mozIndexedDB ||
                   window.webkitIndexedDB;

window.IDBKeyRange = window.IDBKeyRange ||
                     window.mozIDBKeyRange ||
                     window.webkitIDBKeyRange;
```