HTML5 Offline Technologies

Andrea Wharry
College of Science and Technology – CPS 596M
Central Michigan University
Mount Pleasant, MI United States
May 2, 2014
Wharr1ar@cmich.edu

Abstract — **This research paper investigates the enormous potential of the existing and immensely underutilized HTML5 offline technologies. The paper will also expand on its uses for mobility in an offline environment. The importance of offline accessibility and user experience will be explored, in addition to supporting data for why integrating offline technologies with web applications is advantageous. Information will be included as to why adoption is necessary to accommodate modern day users and infrastructures. HTML5 offline technologies are vast, useful and provide web applications with different offline solutions within its APIs. The article will expand on several APIs available through HTML5 and the practical uses for each. The necessary steps taken for adoption will be covered, including file attributes and general architecture of the HTML5 offline implementation. The main web storage and HTML5 application cache prototype will then be discussed, along with the challenges and benefits of using the particular specifications in the development and how they differ from the current methods. The research will also touch on reasons why the HTML5 offline web tools may be underutilized, elaborate on some of the major fundamentals that are missing to accommodate these technologies, and possibilities for future applications of HTML5 offline technologies.**

Keywords — HTML5; application cache; web storage; mobile computing; user experience; PhoneGap.

IMPORTANCE OF OFFLINE TECHOLOGIES

Technological integration within society has produced a demand for methods that offer a seamless flow of data. With mobility in mind, it has become a challenge for developers to provide their online web applications with offline accessibility. A common scenario used to describe the challenges of offline accessibility is mobile device usage in flight or during routine website maintenance. A March 2014 survey conducted by the Smart Insights marketing agency found that consumers greatly preferred the use of mobile apps to their mobile browser for consuming web media [1]. It was observed that 89% of a user's time spent viewing media content was through a mobile application, while only 11% was with the web browser [1]. This is clear evidence that web content must be optimized for use outside of a generic browser. Users may expect an application to have more functionality offline as opposed to a mobile browser, which makes it beneficial to create and maintain a seamless user experience with web applications. With HTML5 Offline Web Application APIs, content can be developed for a website and still have the offline support that users have come to expect from applications. There are strong implications that proactively integrating offline technologies will help web services improve customer loyalty, extend usability and versatility and improve efficiency over web applications.

HTML5 OFFLINE WEB APPLICATION APIS

With the dramatic growth of mobile app traffic, developers can improve the user experience by accommodating this demand with HTML5 offline web application APIs. Data loss in mobile applications due to a network outage or error is a reality that many users live with on a day to day basis. To most users, this kind of "no guarantee" service is unsatisfactory. This can lead to frustration, distrust and make some customers weary of the product. HTML5 Offline Web Application APIs provide remedies to these unpleasant aspects of the web. The APIs enable users to continue interacting with applications without network connectivity, even allowing for storage of data for later submission when the network becomes available.

There are five offline APIs that will be discussed in this section and a basic description for each kind of storage option.

1. *Application Cache* (See Fig. 2)

Application cache (appcache) is the most widely known of all the HTML5 offline technologies used to date. It consists of a manifest attribute added to the opening <*html*> tag and a manifest file that acts as a harbor for all files or images that are desired to be cached [2].

Appcache is beneficial for all types of web apps. It works well for static websites or for a site's shell. Newsfeeds and other dynamic items are not recommended to be add to the appcache manifest file since their content is always changing.

2. *Web Storage*

The Web Storage (Local Storage or DOM Storage) mechanism is the most widely supported among common browsers. With this type of offline API the data is stored locally in the web browser. This is a faster and more secure alternative to cookies because it retrieves the data only when it is requested. Web storage uses key/value pairs to store the data and each web page can only access its own stored information. And although the information is not transferrable to the server without manual intervention, it provides a seamless user experience for use of web applications [3].

This specification is used for when you need to store large amounts of data for an extended duration (or until the browser cache is manually cleared). Web Storage is also no longer exclusive to HTML5 and can be used as a standalone tool [3].

3. *Web SQL Database**

Considering the name, the Web SQL Database API operates just as one would think and was inspired by early developments of SQLite. The specification provides a wrapper around a SQL database that allows execution of SQL statements through Javascript (using CREATE TABLE, SELECT, UPDATE, INSERT, DELETE etc.) and is stored for later retrieval [5].

4. *Indexed Database*

Indexed Database (IndexedDB) is a hybrid API solution that resembles the attributes of Web Storage and Web SQL Databases. As opposed to a relational database, IndexedDB uses an 'object store' that can be used to send and retrieve data through Javascript Objects [9]. Method calls on the database object replace the use of SQL for interacting with the information. The Javascript objects are categorized by a common key path attribute, and data lookup is achieved through matching key/value pairs. The database also utilizes a transaction object to track changes to the data and synchronize changes. The majority of IndexedDB implementations use the Asynchronous API, which uses a callback function for data retrieval instead of return values from SQL queries [4]. This setup allows for database operations to run in the background, but also result in more delays if a request fails [3].

5. *File Systems**

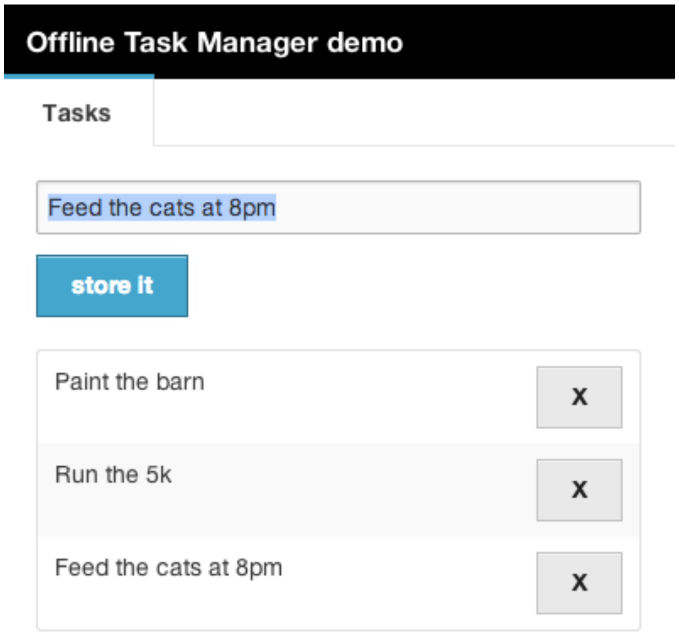
The File System API specification is designed to read, write files, navigate and create directories for storage purposes [7]. On April 24th 2014, the World Wide Web Consortium declared that File System specification as 'discontinued' and is no longer maintained [6]. This API has the least amount of support among the most widely used browsers.

*Note: These offline APIs are no longer being maintained and have limited support [5][7].

PROTOTYPE SPECIFICATIONS AND IMPLEMENTATION

The web app prototype is designed as a proof of concept for methods of offline storage and supports the existing implementation tools provided by HTML5. The project utilizes two different types of local storage techniques touched on in the previous section: Application Cache and Web Storage. The example built to demonstrate offline availability is a task manager list, depicted in figure 2, which allows a user to enter any number of tasks to a list on the web page that can be saved and referenced at a later time. Each new entry from the user is stored locally with Web Storage. This allows a user to add or delete tasks from the list and maintains the correct structure with or without network connectivity. The Application Cache handles caching and serving files for offline viewing and consistency while the Web Storage handles storing the entries and logging offline submissions.

Fig. 1: Interface of the task manager app.



A. *APPLICATION CACHE STRUCTURE AND ARCHITECTURE*

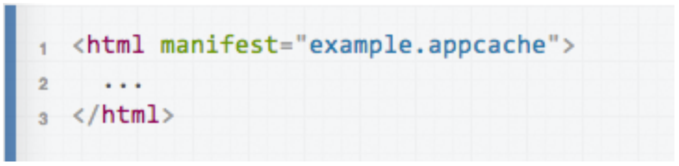
As mentioned in a previous section, the structure for Application Cache consists of adding a manifest attribute to every page that must be cached, as well as the inclusion of a main manifest file [8]. The manifest file is only a text file, and it is conventional to format it with the file extension ".appcache" and adding it to your root directory. In truth, so long as the same relative path to the file is included in the manifest attribute for each HTML page, it does not matter what the extension is. Figure 2 shows an example of the `manifest="example.appcache"` attribute that the developer must include to ensure that this particular page is selected for caching. If this attribute is not included in the HTML file, then the page will not be cached even if this particular page is added in the main manifest file. If HTML files are referenced in the main manifest file that do not contain the above attribute, then all other files in the manifest will not be cached [11]. This occurs even if the other files have correctly linked manifest attributes.

Assuming none of the above errors are true, the files included in the manifest will now be accessible both online and offline. Upon first visit to a page that declares a manifest, the browser will try and update the cache [11]. During an update, all previous browser cache is replaced with the new cache. The events (caching, checking, updating, and progress etc.) are achieved by customizable Javascript. The `'window.applicationCache'` object provides events and properties that handle data retrieval. The following are the events and descriptions of the app cache life cycle:

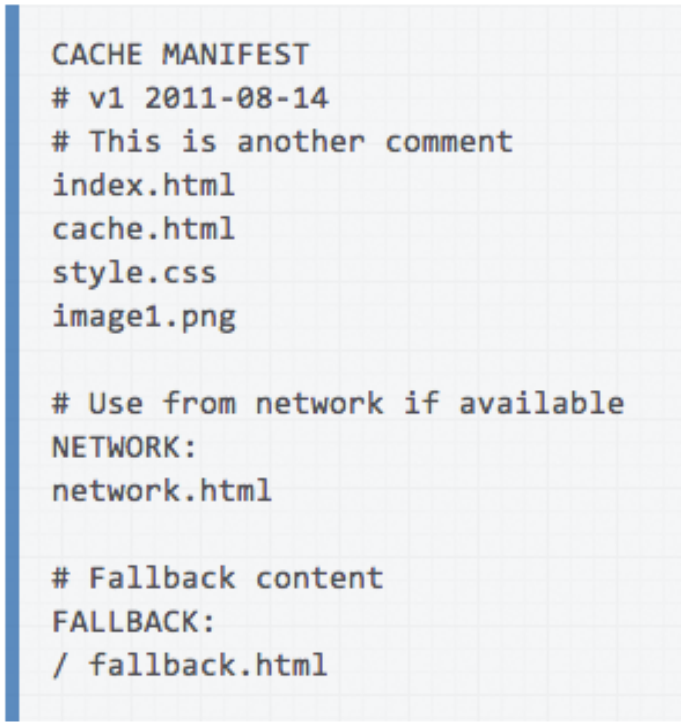
- `checking` – only occurs during the first retrieval of data [2].

- `noupdate` – "ok" status when no changes are needed [2].
- `downloading` – checks the manifest file for an updated version [2].
- `progress` – indicates the number of files processed and number of files to be downloaded [2].
- `cached` – indicates the resources in the manifest have been downloaded and are now cached [2].
- `update ready` – indicates the old cache can be switched with the updated files [2].
- `obsolete` – occurs when no subsequent manifest file was found [2].
- `error` – handles errors when interacting with the manifest file [2].

Fig. 2: Inline manifest file declaration (top). The basic contents of the manifest file (below).



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



```
CACHE MANIFEST
# v1 2011-08-14
# This is another comment
index.html
cache.html
style.css
image1.png

# Use from network if available
NETWORK:
network.html

# Fallback content
FALLBACK:
/ fallback.html
```

B. *Web Storage API*

Similar to cookies, the Web Storage API is an object based storage interface that allows key/value pairs to be stored locally for faster lookup and access of information [3]. Unlike cookies it is far less taxing on bandwidth since it does not require retransmission to the remote web server. Another benefit of Web Storage is that it is capable of storing significantly more data locally. The Web Storage API has two types of storage for web data, localStorage and sessionStorage. The data stored in the localStorage object will not expire unless explicitly removed or cleared from the cache [10]. Because it is stored on the client side, website data can be retained even after navigating away from a page or closing the browser entirely. SessionStorage does exactly as its name suggests and only keeps local data until that particular session is complete. It is commonly used to save user data between tabs, such as login credentials. In regards to actual data access, Web Storage is similar to the indexedDB [10]. Queries for data are submitted by calling the methods for the localStorage or sessionStorage object. Since Web Storage is natively included with HTML5, it is supported in most modern browsers.

One of the drawbacks to this specification is data deletion. If an item needs to be removed from the database, it must be removed manually due to the permanency of localStorage [10]. This can make data management cumbersome when working with limited local storage. Another drawback for Web Storage is the data structure. Since Web Storage is object-based, queries on the storage object are less efficient than a regular table lookup [10]. Lastly, localStorage may be a weak spot in data security if it stores sensitive information like passwords on the client side [10].

How Current Caching Methods Differ

While it is common for most browsers to cache data locally for optimization purposes, older caching methods gave the client little or no control over what can stay in cache and what will be overwritten [8]. In addition, browser caching does not provide a reliable way for users to access pages or work offline. App Cache is able to address some of these issues and improve on other aspects of user experience with the mobile web as well. By keeping a manifest file in sync with the web server, App Cache can drastically improve load times. The manifest file also allows the content creator to specify what aspects of the website are optimal for caching in order to work in an offline environment. App Cache is able to extend workability to offline, by accepting user input and then with the help of Web Storage, it holds the data in locally whether the network is restored or not [8].

Another method of local storage utilized by browsers are cookies. Similar to caching, cookies contain small amounts of web data in order to improve site load times and store commonly accessed information such as usernames or the contents of an online shopping cart. Session cookies will temporarily store site data until the browser is closed, while persistent cookies can hold web data even while offline [10]. Cookies regularly transmit data to the web server and can cause longer load times. The storage capacity of a cookie is also low, limiting the type of content that can be saved locally [10]. The Web Storage API, as discussed, is able to solve some of these issues with session, local, and object-based storage. Web Storage has similar functionality to cookies but is able to hold significantly more data, making local storage much more feasible of an option [10]. The sessionStorage object functions similarly to session cookies, wherein it only holds data until the web browser has been closed. LocalStorage is the Web Storage equivalent of persistent cookies, but requires more space on the client side.

Application Cache Benefits and Drawbacks

The primary benefit of utilizing the Application Cache specification is increase information availability, allowing users to visit and navigate a site whether there is an active network connection or not. In addition, cached resources allow for faster load times since a specified portion of the page is available in local memory [11]. The browser can load the local content much faster and the web server only needs to retrieve content updates. Since the browser is only required to download the resources that have changed since the last time of access, this can also reduce the server load.

One major drawback associated with the API is browser compatibility. There are a number of web browsers that are not currently compliant with the Application Cache API that may result in a negative impact on end users being unable to use it with their preferred browser. Most browsers also limit memory for caching files. Depending on the number of sites utilizing Application Cache on a single device, limitations on storage may prevent it from being fully utilized. On average, peak caching limitations are about 5 Mb but this may change over time [10]. Lastly, there are some issues when debugging. Even with common "gotchas" well documented, these issues can result in wasted time and unnecessary frustration for the developer. A few relevant "gotchas" encountered in this project were:

1. *Apps only updates if the manifest file changes* – this is resolved by adding a version at the top of the file, but this can slow down development.
2. *Items not included in manifest will not load in a cached page* – there are no options for images set in the CSS file. These files must be added to the manifest.
3. *No more conditionals* – this includes responsive designs and can be inconvenient when developing for mobile-friendly versions.
4. *If one CACHE file can't be retrieved, the entire cache will be ignored* – this is an "all or nothing" approach and is counter-productive.

Utilization of HTML5 Offline Technologies

From inception, HTML5 offline APIs have had varying adoption rates by browsers and developers worldwide. According to statistical research of module usages by the content management system (CMS) company Drupal, the HTML5 offline application cache module has not been widely disseminated among their clients [19]. As of April 20th 2014, only 50 out of 1,006,526 sites powered by Drupal are utilizing the HTML5 offline module [18][19]. Although the severe lack of adoption may have other key factors playing a role in the underutilization, it exemplifies the similar acceptance rate across the web [2]. CMS or not, this may be due to HTML5's 'candidate recommendation' status, meaning that it is still undergoing development. HTML5 as a standard is not expected to have full 'recommendation' status until the end of 2014, so many developers may be slow to adopt in order to tread lightly. Despite its lack of full recommendation, many popular web browsers have enabled support for HTML offline technologies. *(See Table I)*

TABLE I: Compatible Browsers

Common Browsers	*Application Cache (82%) [13]*	*Web Storage (89%) [15]*	*Web SQL Database (56%) [14]*	*Indexed Database (65%) [12]*	*File System (40%) [16]*
IE	10.0+	8.0+	Not Support	10.0+	Not Support
Chrome	31.0+	31.0+	31.0+	31.0+	31.0+ (webkit)
Firefox	28.0+	28.0+	Not Support	28.0+	Not Support
Opera	20.0+	20.0+	20.0+	20.0+	20.0+ (webkit)
Safari	5.1+	5.1+	5.1+	Not Support	Not Support
Android Browser	2.1+	2.1+	2.1+	4.4	Not Support
iOS Safari	3.2+	3.2+	3.2+	Not Support	Not Support
Blackberry Browser	7.0+	7.0+	7.0+	10.0 (webkit)	10.0 (webkit)
IE Mobile	10.0	10.0	Not Support	10.0	Not Support
Opera Mini	Not Supported	Not Supported	Not Support	Not Support	Not Support

Table I: This chart displays the percentage of support among browsers and the verisions compatible with the associated APIs. Some browsers provide support for future version as well.

Conclusion and Future Work

To optimize mobility, user experience, and development time, PhoneGap is a feasible option for future implementation. This cross-platform mobile app development framework allows developers to write native applications for various mobile phone platforms using HTML, CSS, and Javascript [17]. Since PhoneGap helps developers focus on a single set of web tools, HTML5 offline technologies are a viable option. Developing with phone gap can drastically cut down on development time and create continuity whether the site is accessed from a web browser or a mobile app. Building with PhoneGap also enables access to native OS functionality using Javascript – this includes utilities such as the phone's camera and GPS [17]. Future development of this project will implement a memory game that keeps track of a user's score and can be accessed offline.

There is no simple solution for improving offline accessibility, but HTML5 offline technologies are a step forward in allowing users to access apps wherever they are, regardless of connectivity. Accompanied with PhoneGap, HTML5 offline technologies offer continuity and allow user access offline.

REFERENCES

[1] D. Bosomworth. (2014, March 24). *Mobile Marketing Statistics 2014.* [Online]. Available: http://www.smartinsights.com/mobile marketing/mobile-marketing-analytics/mobile-marketing-statistics/

[2] W3C. (2011). *HTML5: Offline Web applications.* [Online]. Available: http://www.w3.org/TR/2011/WD-html5-20110525/offline.html

[3] W3C. (2013). *Web Storage.* [Online]. Available: http://www.w3.org/TR/webstorage/

[4] W3C. (2013). *Indexed Database API.* [Online]. Available: http://www.w3.org/TR/IndexedDB/

[5] W3C. (2010). *Web SQL Database.* [Online]. Available: http://www.w3.org/TR/webdatabase/

[6] W3C. (2014). *Fileapi directories and system/filewriter.* [Online]. Available: http://lists.w3.org/Archives/Public/public-webapps/2014AprJun/0010.html

[7] W3C. (2014). *File API: Directories and System.* [Online]. Available: http://www.w3.org/TR/file-system-api/

[8] (2014, April 14). *Using the Application cache.* [Online]. Available: https://developer.mozilla.org/en-US/docs/HTML/Using_the_application_cache

[9] (2014, April 15). *IndexedDB*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API

[10] M. Pilgrim. (2013). *The Past, Present & Future of Local Storage for Web Applications.* [Online]. Available: http://diveintohtml5.info/storage.html

[11] M. Pilgrim. (2013). *Dive In: Let's Take This Offline.* [Online]. Available: http://diveintohtml5.info/offline.html

[12] StatCounter GlobalStats. (April, 2014). *IndexedDB*. [Online]. Available: http://caniuse.com/#search=indexedDB

[13] StatCounter GlobalStats. (April, 2014). *Offline web applications* [Online]. Available: http://caniuse.com/#search=offline%20web%20applications

[14] StatCounter GlobalStats. (April, 2014). *Web SQL Database* [Online]. Available: http://caniuse.com/#search=Web%20SQL%20Database

[15] StatCounter GlobalStats. (April, 2014). *Web Storage – name/value pairs.* [Online]. Available: http://caniuse.com/#search=Web%20Storage

[16] StatCounter GlobalStats. (April, 2014). *FileSystem & FileWriter API.* [Online]. Available: http://caniuse.com/#search=FileSystem

[17] PhoneGap (2014). *About PhoneGap* [Online]. Available: http://phonegap.com/about/

[18] Drupal (April, 2014). *Usage statistics for Drupal core.* [Online]. Available: https://drupal.org/project/usage/drupal

[19] Drupal (April, 2014). *Usage statistics for HTML5 Application Cache.* [Online]. Available: https://drupal.org/project/usage/appcache