TATA CONSULTANCY SERVICES
Experience certainty.

White Paper

Developing Offline Web Applications using HTML5

Building the applications of tomorrow with rich features requires proper understanding of the key Web standards and frameworks that are emerging today such as HTML5, CSS3, JavaScript, and jQuery. Among all the latest technologies, HTML 5 is one of the most- widely accepted technologies, which offers Dynamic user experience, responsiveness, and seamless usability across different browsers and devices with a lot of attractive features.

New features of HTML5 include a set of elements along with the support for new Application Programming Interface (APIs). HTML5 provides APIs like GeoLocation, Webstorage, indexDB and so on. From the list of APIs, the developers can program their Web applications as per their requirement and the situation that best fits to the scenario. Here in this white paper, we will discuss the development of Web applications that enables the user to have the experience of the Web, in offline mode along with its advantages and limitations.

About the Author

Nilachala Panigrahy

The author, Nilachala Panigrahy, is a Systems Engineer, working with TCS. He has more than two years of experience in developing and managing Rich Internet Application (RIA) for various domains. He also has the expertise in User Experience space. His competency includes RIA technologies like Flex, jQuery, GWT, and HTML5. He has worked on different RIA Web applications and has helped in making the websites browser compatible and usable.

Table of Contents

Introduction

HTML 5 is the most recent version of Hyper Text Markup Language (HTML), a language proposed by Opera Software for Web (World Wide Web) content presentation. HTML is in the process of continual development since the day it was created in early 1990s and HTML5 is the fifth version of HTML standard. HTML5 covers the features of HTML4, XHTML 1 and DOM2HTML. It is a formatting language that programmers and developers use to create documents on the Web.

With a number of new elements, attributes, 2D and 3D graphics, video, audio elements, localStoragefacility, local SQL database support and a lot more exciting features, HTML 5 has brought a monumental change in World Wide Web. HTML 5 is nothing but using shorthand for continuous innovation in a client-centered application with new tags on a general development framework with CSS3 and JavaScript. It supports both desktop deployment and mobile deployment. Smart phones like Apple iPhone, Google Android, and phones running Palm WebOS have gained huge popularity with HTML 5 based rich Web applications. HTML 5 contains a number of new and easily understood elements in various areas. Some of these are as follows:

- HTML controls used in UI
- Multimedia file supports like video and audio
- Database support like local SQL database
- A number of new Application Programming Interfaces (APIs)

These elements can be used in interactive websites with little customization to add an attractive effect and enhance the performance.

Apart from the above features, the beauty of HTML5 comes with the fact that it supports the development of Web applications that can be used in offline mode when there is not a prolonged and constant access to the internet due to usage policies like social networking APIs (such as Facebook, Twitter and so on).To store the data locally and to allow the applications to run offline, HTML5 provides three different APIs and these are:

- Web storage: Can be used for basic localstorage with key-value pairs
- Offline storage: Can be used to save files for offline use
- indexDB: Supports relational database storage

Webstorage API

To overcome the limitations of cookies, Webstorage APIs are being introduced in HTML5 that can be used to store the data locally on the user's machine. This API allows the developers to store the data in the form of KEY-VALUE pairs that can be accessed by the Web applications through scripting. Web storage works exclusively with client side scripting where data can be transmitted to the server on requirement basis and a Web server cannot write the data to Web storage as well.

Web storage provides two different storage areas that differ in scope and lifetime and those are:

- LocalStorage
- SessionStorage
- Both "localStorage"and "sessionStorage" objects provide certain useful properties and methods such as:

getItem() Method	: Get the value of item for the key being passed as the parameter
length Property	: Return the length of the number of items
remainingSpace Property	: Return the remaining storage space in bytes, for the storage object
clear() Method	: Remove all the key/value pairs from Web Storage
key() Method	: Retrieve the key at specified index
removeItem() Method	: Remove the specified key/value pair from Web Storage
setItem() Method	: Set a key/value pair

Local Storage

The localStorage object is useful when one wants to store data that spans across multiple windows and persists beyond the current session. This object provides persistent storage area for domains. The localStorage keeps the data between browser sessions, even after the browser is closed and reopened.

```
if(window.localStorage){
                if (localStorage.pagecount)
                {
                        localStorage.pagecount=Number(localStorage.pagecount) +1;
                }
                else
                {
                        localStorage.pagecount=1;
                }

                document.write("You are viewing this page  " + localStorage.pagecount + "
                                                time(s).");
        }
        else{
                alert("lack of support for localStorage");
        }
```

In the above code, first the browser compatibility is checked. If it found compatible, then we perform the storage related activities, else the user is informed about the browser incompatibility. If the browser supports Webstorage, then we are maintaining a count that keeps on increasing whenever the user either refreshes the browser or reloads after closing the window.

Benefits of localStorage object are:

- Long term data persistence
- Absence of HTTP overhead of cookies
- Effective user storing preferences
- localStorage objects spanning multiple windows and persisting beyond the current session.

SessionStorage

The sessionStorage object stores the data for one session. The data is deleted when the user closes the browser window. It allows separate instances of the same Web application to run in different windows without interfering with each other. Once, the browser's session ends (window/tab closed), sessionStorage ends .Values put into sessionStorage are only visible in the window/tab that created them .Opening page in new window or tab starts a new session.

```
if(window.sessionStorage){
                if (sessionStorage.pagecount)
                {
                        sessionStorage.pagecount=Number(sessionStorage.pagecount) +1;
                }
                else
                {
                        sessionStorage.pagecount=1;
                }
                document.write("You are viewing this page " + sessionStorage.pagecount + "
                                                time(s) during this session.");
        }
        else{
                alert("lack of support for sessionStorage");
        }
```

Storing and Retrieving Data

Any type of data can be stored and retrieved form localStorage using setItem() and getItem() methods of the localStorage object. Everything is stored in the form of string within the storage area. It is necessary to convert the data into a string before storing the data into the storage area. While retrieving the data from the localStorage, we need to parse the data before putting that into appropriate use, if the data is other then String type.

```
function storeData(){
                        eid = document.getElementById("txtEmpID").value ;
                        ename = document.getElementById("txtEmpName").value ;
                        designation = document.getElementById("ddDesignation").value ;

                        if(window.localStorage){
                                    localStorage.setItem("empID",eid);
                                    localStorage.setItem("empName",ename);
                                    localStorage.setItem("empDesignation",designation);
                        }
                        else
                        {
                                    alert("lack of support for localStorage");
                        }
            }
```

The above code first retrieves the values from different fields and then stores that into the storage area using the setItem() metod that accepts a key name("empID") and a corresponding value (value of the variable "eid") for that.

```
function retriveData(){
                        eid = localStorage.getItem("empID");
                        ename = localStorage.getItem("empName");
                        designation = localStorage.getItem("empDesignation");

                        document.getElementById("txtEmpID").value = eid ;
                        document.getElementById("txtEmpName").value = ename ;
                        document.getElementById("ddDesignation").value = designation ;

            }
```

The above code first retrieves the values from the data storage using the respective key names with the help of the getItem() method that accepts the key name as a parameter and stores the retrieved value in a variable. The value of the variables are then used to populate the fields of the Web page.

Clearing the Data

Data stored using the storage objects can be deleted by any one the following methods:

- Using browser options
- Using the clear ( ) method form the application

The following code is used to delete all the contents form the storage area when the user clicks the "clear" button

```
function clearData(){
        localStorage.clear();
}
```

However, if the user wants to delete a specific item form the storage, then he can use the removeItem() method with the key name as a parameter that needs to be deleted .

Handling Changes

Any changes done to the storage area can be tracked by adding an event listener into our code.

The event gets triggered on the window object whenever the data in the storage area actually changes. The event is never triggered in the session that initiated it but gets fired for other tabs or the sessions.

```
if(window.addEventListener){
        window.addEventListener("storage",change_tracked,false);
}
else{
        window.attachEvent("onstorage",change_tracked);
}

function change_tracked(e) {

        if(e.key == "empName") {
                alert('The value for ' + e.key + ' was changed from' + e.oldValue + ' to ' +
                                                        e.newValue);
        }
}
```

The above code adds an event listener that calls the "change tracked" method asynchronously whenever any changes are done to the storage area. "change tracked()" method keeps the track of the changes done to the value of the key "empName". If any changes are done, then it alerts the user with the changes.

Use Cases

- Storing simple data that needs to remain persistent after the user has browsed away from the application or closed the Web browser
- Saving game state, navigation location, or storing some specific information that can be used across the entire Web application
- Intimating the user about the changes done (for example, account no.) if the same page is opened in multiple tabs

HTTP Cookies versus Webstorage

- HTTP Cookies suffer from limited data storage (e.g. 4KB), whereas the data storage capacity of Webstorage API for most of the browsers is 5MB as per w3c specification with an exception to IE8 (and upwards) that supports up to 10MB.
- Cookies limit the accessibility of data to a certain domain name or a URL path where as Webstorage can limit the access to a certain domain name, or to domain TLD (like .org) or for the given user session.
- It is possible to iterate through all the key/value pairs in HTTP cookies, whereas in Webstorage, it is not possible to iterate through keys unless the key is known.
- Data stored in cookies are passed on by EVERY request to the server, making it very slow and in-effective. However, the Webstorage API data is NOT passed on by every server request and used ONLY when asked.
- Webstorage is more public then cookies. This is the reason why we need to take special precautions to ensure the security.

Offline Web application API

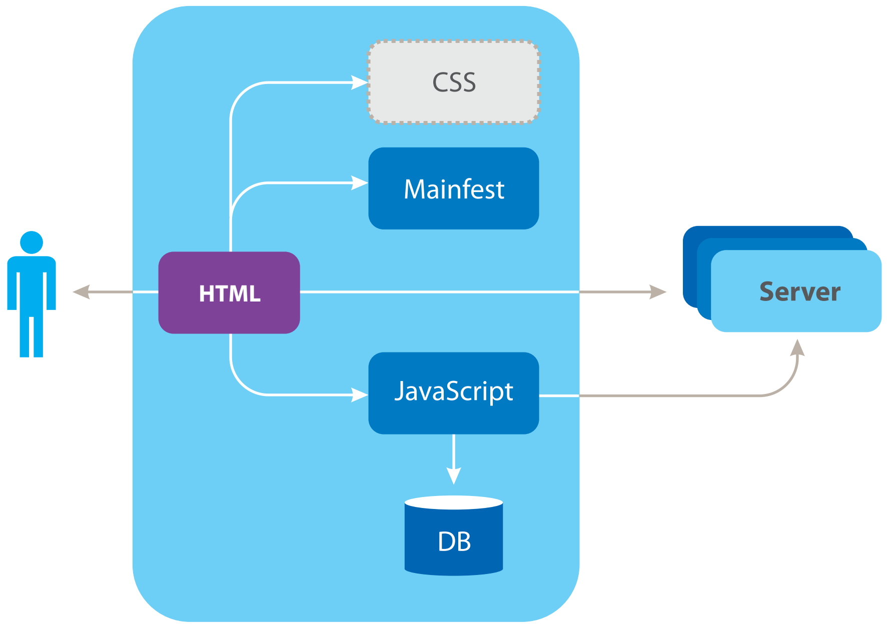
To give the user the experience of the seamless Web browsing, most of the browsers provide the facility of caching. But caching suffers from the following limitations:

- Developers do not have any control over the pages to be cached. Hence they cannot cache all the files of any specific application.
- It does not provide a reliable way for you to access pages while you are in offline mode (for example, Airplanes).

To overcome the above limitations, HTML5 comes up with the concept of Offline Web application API (AppCache) which provides the following features:

- The user can use cloud functions on a mobile device, work offline with a locally deployed application on a local database, and share data with the rest of the cloud when going online again.
- AppCache caches only the specified pages that are included in the manifest file and hence it is reliable.
- With a higher degree of certainty, it gives the user the control of the way the Web applications should look when offline.

The following figure shows the major components of an offline Web application.



Architecture of an Offline Web Application

HTML pages
Traditional Web application consists of HTML pages that contain the displayed data and the render information. Whenever a user initiates an event, it causes a request-response cycle with a page load and the execution of associated JavaScript functions.

Offline Web application consists of a single HTML page without the need for loading of further HTML pages through the request-response cycles. The whole action is on one page.

JavaScript
These files contain the functions that are useful for handling the events initiated by a user on the HTML page.

Cascading Style Sheet (CSS)
It describes the way HTML page should be rendered. For mobile devices, there are various JavaScript/CSS libraries and frameworks to deliver a near native user experience with Web applications (for example, iUi for the iPhone).

Database
The HTML5 standard introduced local database storage. It is implemented in current versions of the Apple® Safari browser. The browser provides an embedded database, with SQLite, that can be accessed from the JavaScript by processing SQL queries. The business data of the application model is stored here.

Manifest
The manifest file is the mandatory deployment descriptor component for an offline Web application. It simply lists all the files that need to be loaded.

Steps to Follow for Creating Offline Web App

Checking Browser Compatibility

Due to the variation in browser support, first we should check the compatibility of the browser before availing an application offline.

It can be done in two ways:

- Without using Modernizer Object
- Using the Modernizer Object

Without using the Modernizer object, we can check the compatibility in the following manner:

```
function chk_offline_compatibility(){
        if(window.applicationCache) {
                alert('this browser supports offline Web apps' );
        }
        else {
                alert('sorry your browser doesn't support offline Web app' );
        }
}
```

The above code checks the global window object to see if it supports the application Cache object or not and informs the user about the status accordingly.

Using the Modernizer object, one can check the compatibility in the following manner:

```
if (Modernizr.applicationcache){
                alert('this browser supports offline Web apps' );
        }
        else {
                alert('sorry your browser doesn't support offline Web app' );
        }
```

Benefit of using the Modernizer object is that it can handle certain tricky marginal cases. For example, in private browsing modes, such as Chrome's incognito mode, a call to windowapplication Cache will return true, however the browser will not actually be able to write files to the cache.

Creating a manifest file

This file resides on the server and decides which files should be stored client-side in the browser's AppCache, that can be used when the user goes offline .This file contains a list of URLs to various resources such as Javascript files, HTML CSS, images and flashfiles. The manifest file has a basic structure that should start with CACHE MANIFEST.Comments can be made by putting # at the beginning of a line. File names must be listed exactly as they appear on disk as it is case sensitive.

There are three namespaces that can be included multiple times in a manifest file and those are:

CACHE

NETWORK

FALLBACK

Cache

This section lists all the files that the browser needs to cache for offline access. The paths provided can be either relative or absolute. Inclusion of this header is optional as this is the default header but if we want to flag files to be cached anywhere else in the file, we need to place them under an explicit CACHE: header. Only one file name per line is allowed. Wildcards and fragmented identifiers are not allowed under this header.

A sample of the cache section that loads html, css js files along with few images looks like this:

```
CACHE MANIFEST
style.css
offlinescript.js
images/image1.png
images/image2.jpg
```

Files listed in this section are never loaded from the server even when the user is connected to the internet. They are always loaded from the AppCache.

Network

This declaration is used to specify files that should not be cached (for example, a login page). Files listed in this section will not be loaded from the application cache, but will be retrieved from the server if the browser is online. We can specify "*" (the default), which sets the online white list wildcard flag to "open", so that resources from other origins will not be blocked.

```
NETWORK:
Login.html
dynamic_script.cgi
```

Whenever the user tries to access the files listed in this section, he or she gets the acknowledgement with an appropriate offline message.

Fallback

When the user tries to access a page that was not cached, or, something that was supposed to be cached but was not saved correctly, the cache manifest API offers a FALLBACK section that points to a page which will be loaded. For example, when the browser is offline and tries to load something that is not in the application cache, such as a page or JavaScript file listed in the NETWORK section, then it can be redirected to an appropriate page listed in this section. The fallback content is cached and used in case the main content does not load.

```
FALLBACK:
offline.html
image1.jpg  alt_image.jpg
/cgi/scripts/  default.html
```

In the above example, "alt_image.jpg" is cached by AppCache. If "image.jpg" cannot load, the "/" represents a wildcard fallback that gets triggered whenever there is a fallback while loading any of the ".cgi" and/or script files.

Using the Application Cache API

To use the files from application cache, we can use the application Cache API.

The "window.applicationCache" object provides events and properties that can be used to deal with data retrieval. Current status of the application cache can be checked using   window.applicationCache.status, which returns a numeric value mapped to the following states:

0 – uncached
If the page is not linked to the application cache; also, the very first time the application cache is being downloaded

1 – idle
When the browser has the latest version of the AppCache, and there are no updated versions to download, then the status is set to "Idle"

2 – checking
The duration of when the page is checking for an updated manifest file, then the status is set to "Checking"

3 – downloading
The duration of when the page is actually downloading a new cache (if an updated manifest file has been detected); the status is set to "Downloading"

4 – update ready

Once the browser finishes downloading that new cache and is ready to be used (but is not being used yet); during this time, the status is set as "Update ready"

5 – obsolete

In case the manifest file cannot be found, then the status is set to obsolete and the application cache gets deleted.

Events

Certain events also get fired, depending on what is going on with the AppCache at the moment.

Checking
This gets fired when browser is attempting to download the manifest for the first time, or is checking if there is an updated version of the manifest file.

Noupdate
If there is no updated version of the manifest file on the server, then "noupdate" is fired.

Downloading
If the browser is downloading the cache for the first time, or if it is downloading an updated cache, then this is fired.

Progress
This is fired for each and every file which is downloaded as part of the AppCache.

Cached
This is fired when all the resources have finished downloading, and application is cached.

Updateready
Once resources have finished re-downloading for an updated cached file, then updateready is called. Once this happens, then we can use swapCache() to make the browser to use this newly updated cache.

Obsolete
This is fired if the manifest file cannot be found (404 error or 410 error).

Error
This can be fired for a number of reasons.

If the manifest file cannot be found, then the application cache download process has to be aborted, in
which case this event can be fired.

It can also be fired in case the manifest file is present, but any of the files mentioned in the manifest file cannot be downloaded properly.

It can even be fired in case the manifest file changes while the update is being run (in which case the browser will wait a while before trying again), or in any other case where there is a fatal error.

window.applicationCache.update(): This will trigger the application cache download process, which is nearly the same as reloading the page. It simply checks if the manifest has changed, and if so downloads a fresh version of all the content in the cache

window.applicationCache.swapCache(): This function tells the browser to start using the new cache data if it is available

Add the manifest file to the local server

Once the .manifest file is created we need to add that file to the root Web directory of our application (in case of Apache) which is done as follows:

```
AddType text/cache-manifest .manifest
```

The above code adds a directive in the ".htaccess" file.

This directive makes sure that every manifest file is served as "text/cache-manifest". If the file is not served as required, the whole manifest will have no effect and the page will not be available offline.

Linking manifest file to html page

The manifest file once created can be added to the html pages with the help of "manifest" attribute of the <html> element.

```
<html manifest="offline.manifest">
```

Once we add the manifest file in our html page we can run it to check the functionality.

At this point of time we need to remember the fact that we need to allow the browser to enable the usage of AppCache, if asked.

Checking for updates

For checking the availability of any updates in the manifest manually, we can write the following code :

```
function onUpdateReady() {
     alert('found new version!');
   }

   window.applicationCache.addEventListener('updateready', onUpdateReady);

   if(window.applicationCache.status === window.applicationCache.UPDATEREADY) {
     onUpdateReady();
   }
```

We can also use the update() to do the same job .

window.applicationCache.update(): This will trigger the application cache download process, which is nearly the same as reloading the page. It simply checks if the manifest has changed, and if so downloads a fresh version of all the content in the cache

window.applicationCache.swapCache(): This function tells the browser to start using the new cache data if it is available

In Chrome we can clear the offline cache by selecting "Clear browsing data..." in the preferences. Safari has a similar "Empty cache" setting in its preferences but a browser restart may also be required.

In Firefox, the offline cache data is stored separately from the Firefox profile—next to the regular disk cache:

Windows Vista/7: C:\Users\<username>\AppData\Local\Mozilla\Firefox\Profiles\<salt>.<profile name>\OfflineCache

Mac/Linux: /Users/<username>/Library/Caches/Firefox/Profiles/<salt>.<profilename>/OfflineCache

In Firefox the offline cache is not cleared via Tools -> Clear Recent History

In case we have changed any of the resources in the AppCache or the offline Web application, then we need to change the cache manifest itself.

While changing the cache manifest, it is recommended to add comments along with the version of change which helps the manifest file in downloading the latest version of the resources.

Browser Support

Offline Web Application API is fully supported by most modern browsers and mobile phones including:

Android 2.0+
iPhone 2.1+
Blackberry 5.0+
Firefox 3.5+
Chrome 5.0+
Opera 10.6+
Safari 4.0+

Use cases

Offline Web Application can be used in the following applications:

- HTML5 offline Web applications can be used in case of devices (mobile) that losses internet connectivity in between.
- It can be used to view and write e-mail message.
- We can also edit documents and presentations offline onboard a flight that does not have Wi-Fi.

indexedDB API

Key-value pairs used by Webstorage API have a limitation of 5MB of storage space and it does not provide efficient searching over values. Even duplicate values can be stored using Webstorage API. To overcome the limitations of the Webstorage API, indexed DB API was being introduced that provides an advanced key-value data management. It is a persistent data store in the browser. indexDB uses an Object Store to store the keys and their corresponding values. It maintains indexes over the records it stores and developers use the IndexedDB JavaScript API to locate records by key or by an index. Each database is scoped by "origin," that is the domain of the site that creates the database.

The Indexed DB specification declares two different API's:

- An asynchronous API, currently implemented by the most modern browsers (IE 10, Chrome 11.0, Firefox 4)
- A synchronous API that is not implemented yet.

Difference between Relational Database and indexedDB

- indexedDB is an Object Store. It is not the same as a Relational Database.
- In a traditional relational data store, we would have a table that stores data in the form of rows where as indexedDB creates an Object Store for a type of data that can store persist Javascript Objects. Each Object Store can have a collection of Indexes that make it efficient to query and iterate across.
- Unlike SQL, it is replaced with a query against an index which produces a cursor that you use to iterate across the result set.

Developing an Application using indexDB

Getting Reference to indexDB

To work with IndexedDB, we need to get a reference. As there is no common specification for achieving it, different browsers support it in their own ways. In Google Chrome, we need to access "window.webkitIndexedDB", while on Firefox, we need to use "window.mozIndexedDB".

```
if ('webkitIndexedDB' in window) {
                window.indexedDB = window.webkitIndexedDB;
                window.IDBTransaction = window.webkitIDBTransaction;
} else if ('mozIndexedDB' in window) {
           window.indexedDB = window.mozIndexedDB;
}
if ( !window.indexedDB ) {
           alert("Sorry your browser does not support indexDB");
}
```

Creating a database

Once you have access to indexedDB reference, we can create (or open) the database. Creating a database is done by calling the open() method of the indexedDB object.

```
if (window.indexedDB) {
            dbRequest = window.indexedDB.open( "DemoIDB", "An object store for
                                                    demo purpose .");
            dbRequest.onerror = errorStatus;
            dbRequest.addEventListener('success', function(e) {
                                    dDemoDB = dbRequest.result || e.result;
                                    displayRecords(e);
                                 }, false);
}
```

In the above code, "DemoDB" is the ID for the newly created database while "An object store for demo purpose" represents the short description of the database. Every command is executed asynchronously and generates a request object with the help of which, we can handle the events 'onsuccess' or 'onerror'.

Creating an object store

Object stores are like tables form relational databases. A database can contain multiple object stores and each store is a collection of records where each record is a simple key/value pair. The records in an object store are automatically sorted in ascending order by keys. Once the database is created, we need to create an object source as follows:

```
function createObjStore() {
   var request = demoDB.setVersion('1.0');
   request.onerror = errorStatus;
   request.onsuccess = function(e) {
   if (!demoDB.objectStoreNames.contains('demoObjectStore')) {
      try {
            var objectStore = demoDB.createObjectStore('demoObjectStore', null);
            dbLog.innerHTML = "<p>Object store created.</p>" ;
      } catch (err) {
            dbLog.innerHTML =  err.toString() ;
      }
}
else
{
            dbLog.innerHTML = '<p> Object store already exists. </p>';
 }
   }
}
```

SetVersion is the place where we can alter the structure of the database. In this place, we can create and delete Object Stores and build and remove indexes. A call to setVersion returns an IDBRequest object where we can attach our callbacks. When successful, we start to create our Object Stores. Object Stores are created with a single call to createObjectStore () method. The method takes a name of the store ("demoObjectStore") and a parameter object.

Version number changes must be performed under the context of a "version change" transaction.
A transaction represents an atomic and durable set of data access. All data that are read or written to the database will happen using a transaction. Each transaction has a mode which declares what type of interactions is allowed within the transaction. This mode is set when creating the transaction and remains the same for the lifetime of the transaction.

The available modes are:

READ_ONLY

READ_WRITE

VERSION_CHANGE

Adding data to object store

Once the data source is created, we can store data into it as follows:

```
function addRecord() {
            var objectStore = demoDB.transaction(["demoObjectStore"],
IDBTransaction.READ_WRITE).objectStore("demoObjectStore");
            var request = objectStore.put(
                        document.getElementById('txtEmpName').value,
                        document.getElementById('txtEmpID').value,
                        document.getElementById('txtSal').value);
                         request.onerror = errorStatus;
                         request.onsuccess = displayRecords;
}
```

Once we are ready with the Object Store, we can add data to it using the put() method. When the call to put is successful, the 'onsuccess' event ("displayrecords") is called which can display the data .

```
        function displayRecords(e) {
          var html = [];
          var transaction = demoDB.transaction(['demoObjectStore'],
                                               IDBTransaction.READ_ONLY);
          var request = transaction.objectStore('demoObjectStore').openCursor();

           request.onsuccess = function(e) {
           var cursor = request.result || e.result;
            if (!cursor) {
                    dbResultWrapper.innerHTML =  '<ul  id="idb-results">' + html.join('') + '</ul>';
                                  return;
          }

     html.push('<li><span class="keyval">', cursor.key,  '</span> ',
                      '| <span class="keyval" >', cursor.value, '</span> ',
                      '<a href="javascript:void(0)" onclick="indexedDbDemo.deleteKey( \'',
                      cursor.key,  '\')">[X]</a></li>');
                                           cursor. continue();
   }
   request.onerror = errorStatus;
  }
```

In the above code, we have first gained the access to the object source with READ_ONLY mode. Once we have access to the object store, we get a reference to the data existing with the help of openCursor() method. With the help of the cursor we can iterate multiple records in the object store. The cursor also has a position which indicates the object it is currently working with and the direction where to the cursor is moving.

Deleting data from an object store

Data stored in an object store can be deleted as well with the help of the key that uniquely identifies that record; this can be done as follows:

```
function deleteRecord(key) {
        var transaction = demoDB.transaction(["demoObjectStore"],
                                     IDBTransaction.READ_WRITE);
        var objectStore = transaction.objectStore("demoObjectStore");
        if (objectStore.delete) {
                     var request = objectStore.delete(key);
            } else {
                     var request = objectStore.remove(key);  /
        }
                    request.onerror = errorStatus;
                    request.onsuccess = displayRecords;
}
```

Updating data

Data stored in an object store can be modified with the help of the key as follows:

```
function updateRecordValue(key, element) {
    var transaction = demoDB.transaction(["demoObjectStore"],
                        IDBTransaction.READ_WRITE);
    var objectStore = transaction.objectStore("demoObjectStore");
    var request = objectStore.put(element.textContent, key);
    request.onerror = errorStatus;
    request.onsuccess = displayRecords;
}
```

Removal of object store

Existing object sources can be removed from the database as and when needed.

```
function removeObjStore() {
            var request = demoDB.setVersion("the new version string");
            request.onerror = errorStatus;
            request.onsuccess = function(e) {
             if (demoDB.objectStoreNames.contains('demoObjectStore')) {
              try {
                     if (demoDB.deleteObjectStore) {
                       demoDB.deleteObjectStore('demoObjectStore');
                     } else {
                       demoDB.removeObjectStore('demoObjectStore');
                     }
              }
              catch (err) {            dbLog.innerHTML = + err.toString() ;            }
             } else {
                     dbLog.innerHTML = "<p> Object store doesn't exist. </p>";
             }
            };
      }
```

Browser Support

indexDB API is fully supported by the following modern browsers :

Firefox 4.0+
Chrome 11.0+

Use Cases

Offline Web application can be used in the following applications:

An idle use of indexed API can be the case where a retailer moves in an area that has no support for internet connectivity but still he needs to visit all the stores in that locality. The entire list of information can be stored offline using indexDB which can help him out.

Conclusion

At this point of time, support for the development of offline Web applications under the light of HTMl5 has limited scope and is not supported by most of the browsers available in the market. Different browser providers are following different approaches to achieve the same. This approach is also suffering from the size constraint as per browser implementation. However, going forward, it is expected that these APIs will be supported by almost all the leading browsers so that there will be a unified approach for the implementation and users can get the best out of these.

References

[1] http://www.html5rocks.com/en/tutorials/offline/whats-offline/#toc-web-storage
[2] http://sixrevisions.com/html/introduction-web-storage/
[3] http://viralpatel.net/blogs/2010/10/introduction-html5-domstorage-api-example.html
[4] http://dev.opera.com/articles/view/offline-applications-html5-appcache/
[5] https://developer.mozilla.org/en/Offline_resources_in_Firefox
[6] http://nparashuram.com/trialtool/index.html#example=/IndexedDB//trialtool/moz_indexedDB.html&selected=#saveData&
[7] https://developer.mozilla.org/en/IndexedDB/Using_IndexedDB

TATA CONSULTANCY SERVICES

Experience certainty.

Contact

For more information, contact **ntdg.ux@tcs.com**

Subscribe to TCS White Papers

TCS.com RSS: http://www.tcs.com/rss_feeds/Pages/feed.aspx?f=w

Feedburner: http://feeds2.feedburner.com/tcswhitepapers

About Tata Consultancy Services Ltd (TCS)

Tata Consultancy Services is an IT services, consulting and business solutions organization that delivers real results to global business, ensuring a level of certainty no other firm can match. TCS offers a consulting-led, integrated portfolio of IT and IT-enabled infrastructure, engineering and assurance services. This is delivered through its unique Global Network Delivery Model™, recognized as the benchmark of excellence in software development. A part of the Tata Group, India's largest industrial conglomerate, TCS has a global footprint and is listed on the National Stock Exchange and Bombay Stock Exchange in India.

For more information, visit us at **www.tcs.com**

IT Services
Business Solutions
Outsourcing