Building offline access to websites using HTML5

Israel Hilerio, Ph.D.
Principal Program Manager, Internet Explorer
Microsoft Corporation

Here is what you will learn …

- Benefits of caching, client storage, and offline awareness
- Types of applications that benefit from offline support
- What technologies make this possible?
- Putting it all together

You can make your websites better by caching information locally and making them aware of network connectivity.

Benefits of caching, client storage, and offline awareness

Food Network demo

Offline access, resource pre-fetches, and local object stores

Website benefits

- Handle connectivity failures
- Websites behave more like apps
  - Offline content access
  - Offline content creation
  - Can be used while offline
- Performance improvement

Types of apps that benefit from offline support

Offline Scenarios

Kids Book

- Client access of cached resources
- Locally cached file resource

Games w/ Scores

- Connectivity Awareness
- Locally cached data resources, small data set, no querying or indexing

Image Viewer

- File resource upload to sever

Recipe Book

- Download of data resources
- Locally cached large data resources into fully capable database

Email Client

- Upload data resource

What technologies make this possible?

Offline supporting technologies

- AppCache
- WebStorage
- Offline/Online Events
- File & Blob APIs
- IndexedDB
- Resource synchronization via XMLHttpRequest (XHR) and WebSockets

AppCache Demo

Offline URI resolution, resource caching, and prefetching

HTML 5 AppCache - Offline

First Run

Later Runs

Fetch Info From Network

Cache Info Locally

Fetch Info From Cache

No Network Available

Internet

AppCache

Internet

Usage of AppCache via Manifest file

```
<html manifest="test.appcache">
  <head>…</head>
  <body>
     <img src="logo.png" …>
     </img>
     …
     <video … src="fish.mp4" …>
     </video>
     …
     <img src="images/kid.png" … />
  </body>
</html>
```

HTML File


```
Cache Manifest
#7/20/2011 v3
Cache:
logo.png
Fish.png

Network:
fish.mp4

Fallback:
images/ noImg.png
```

MIME Type: text/cache-manifest

Manifest File

HTML 5 AppCache – Update Flow

Next Run

Fetch Manifest From Network

If diff Create New Cache

Internet

AppCache

Later Runs

Fetch Info From Cache

No Network Available

Internet

Access web resources offline

- Use AppCache to access web resources on the client
- Update your cached resources by simply updating a manifest file.
- Pre-fetch data before you needed to improve performance.

WebStorage Demo

Offline forms completion

Web Storage Usage

```
<script>
var lStorage = window.localStorage;
function init() {
   if (!lStorage.score)
       lStorage.score = 0;
   document.getElementById('c1').innerText = lStorage.score; }
function save() {
   lStorage.score = getGameScore();}
</script>
<body onload="init();">
   <p> Your last score was:
      <span id="c1">some number</span>
   </p>
</body>
```

Store small content locally

- Use DOM Storage to store small amounts of textual information from your website
- Provides an easy to use API for storing information while offline.
- Both technologies allows the storing of name/value pairs on the client with more disk space than cookies

Offline/Online Event Handlers

```
function reportConnectionEvent(e) {
    if (!e) e = window.event;
    if ('online' == e.type) {
        alert('The browser is ONLINE.'); }
    else if ('offline' == e.type) {
        alert('The browser is OFFLINE.'); }
}
window.onload = function () {
    status = navigator.onLine; //retrieve connectivity status
    document.body.ononline = reportConnectionEvent;
    document.body.onoffline = reportConnectionEvent;
}
```

Handling connectivity changes

- Use offline/online events to verify connectivity
- Every time you can are connecting to your server, consider the possibility that it can fail
  - Consider caching information locally before sending it to the server

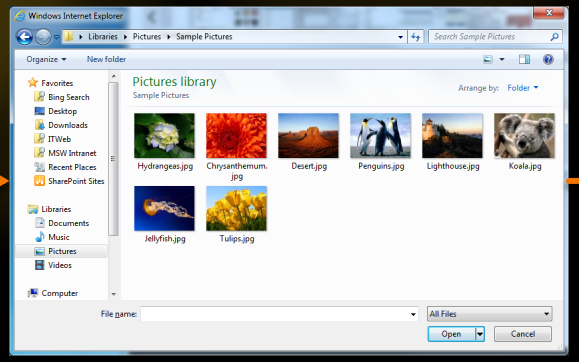
File APIs Demo

Preview local files inside your page without plugins

File Objects

Scenario #1

File System



<input type="file" ...>

File Object (Blob)

Blob URL

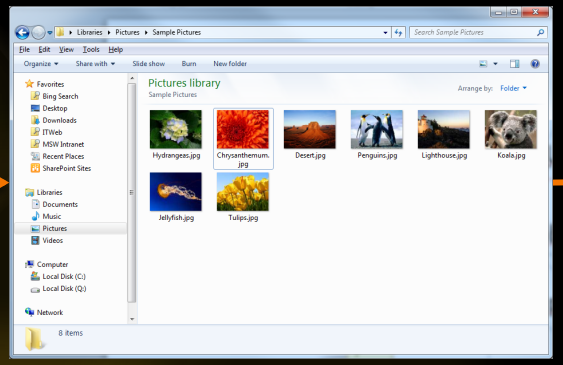
Canvas or Other HTML Element

XMLHttpRequest.send

Web

Scenario #2

File System



Drag & Drop

File Object (Blob)

XMLHttpRequest.send

objectStore.add

Indexed DB

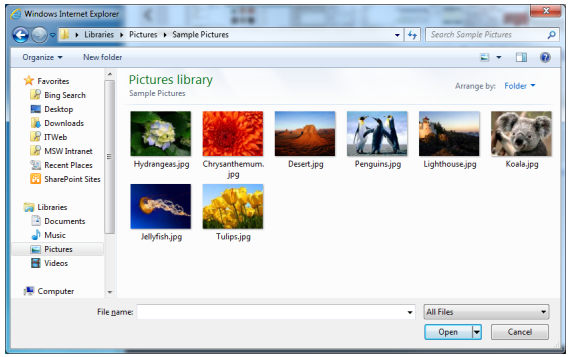
objectStore.get

Previewing Local Resources using File APIs

```
oFReader.onload = function (oFREvent) {
    document.getElementById("uploadPreview").src=oFREvent.target.result;
};

function loadImageFile() {
    if (document.getElementById("uploadImage").files.length === 0)
        { return; }
    var oFile = document.getElementById("uploadImage").files[0];
    if (!rFilter.test(oFile.type))
        { alert("You must select a valid image file!"); return; }
    oFReader.readAsDataURL(oFile);
}
```

```
<input id="uploadImage" type="file" ...>
<img id="uploadPreview" />
```



Read and write local resources

- Use File APIs to read and write local resources without requiring plugins
- Snapshot file information by saving generated Blob into client database
- Map local resources to URIs and load them into any HTML element that supports the mime-type.

IndexedDB Demo

Client Object Store, Indexes, and Filters

Mapping concepts from Relational DB

IndexedDB provides the primitives necessary to build relational query libraries

Concept	Relational DB	IndexedDB (ISAM DB)
Database	Database	Database
Tables	Tables contain columns and rows	ObjectStore contains Javascript objects and keys
Query Mechanism	SQL	Cursor APIs
Joins	SQL	Application code
Transaction Types & Locks	Locks databases, tables, or rows on READ_WRITE Transactions	Lock database on VERSION_CHANGE transaction Lock objectStores on READ_ONLY and READ_WRITE transactions
Transaction Commits	Transaction creation is explicit Default is to rollback unless I call commit	Transaction creation is explicit Default is to commit unless I call abort
Property Lookups	SQL	Indexes are required to query object properties directly
Filters	SQL	KeyRange APIs

ObjectStore access and data loading using IndexedDB

```
var oRequestDB = window.indexedDB.open(“Library”);
oRequestDB.onsuccess = function (event) {
    db1 = oRequestDB.result;
    if (db1.version == 1) {
        txn = db1.transaction([“Books”],IDBTransaction.READ_ONLY);
        var objStoreReq = txn.objectStore(“Books”);
        var request = objStoreReq.get("Book0");
        request.onsuccess = processGet;
    }
};
```

Store, index, and query local data

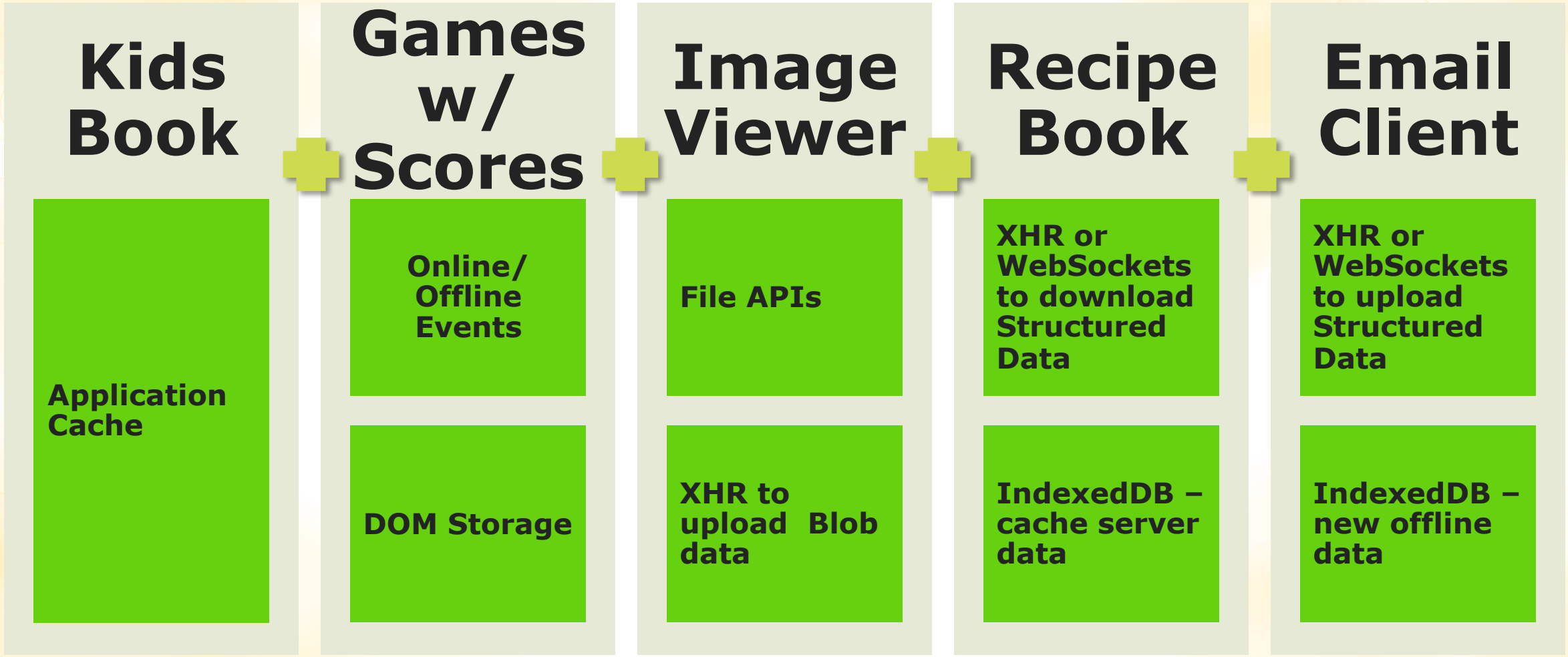
- Use IndexedDB to store, index, and retrieve data on the client
- Cache large amounts of data from the server into the client using objectStores
- Retrieve groups of data using cursors and indexes
- Filter data groups using KeyRanges

Synchronizing content

- Use XHR and WebSockets to upload and download content to and from your server
- Once the content is on the client, the files can be evaluated or parsed to created JS objects that can be stored on the client

Putting it all together

Offline Scenarios



Summary

- There are different scenarios where your apps can be made even better if you leverage these offline technologies.
- Anticipate that your users will want to access information when there is no network connectivity.
- Cache as much information as possible in the client.

Further reading and documentation

Internet Explorer Test Drive site

http://ietestdrive.com/

Internet Explorer Engineering Team Blog

http://blogs.msdn.com/b/ie/

W3C CSS Specifications and Drafts

http://www.w3.org/Style/CSS/

What's new in IE9

http://www.ietestdrive.com/Links/DevGuide9.html

What's new in IE10

http://www.ietestdrive.com/Links/DevGuide10.html

Microsoft®