A beginner's guide to using the application cache

- The cache manifest file
  - Referencing a manifest file
  - Structure of a manifest file
- Updating the cache
  - Cache status
  - AppCache events
- References

It's becoming increasingly important for web-based applications to be accessible offline. Yes, all browsers can cache pages and resources for long periods if told to do so, but the browser can kick individual items out of the cache at any point to make room for other things. HTML5 addresses some of the annoyances of being offline with the ApplicationCache interface. Using the cache interface gives your application three advantages:

1. Offline browsing - users can navigate your full site when they're offline
2. Speed - resources come straight from disk, no trip to the network.
3. Resilience - if your site goes down for "maintenance" (as in, someone accidentally breaks everything), your users will get the offline experience

The Application Cache (or AppCache) allows a developer to specify which files the browser should cache and make available to offline users. Your app will load and work correctly, even if the user presses the refresh button while they're offline.

The cache manifest file

The cache manifest file is a simple text file that lists the resources the browser should cache for offline access.

Referencing a manifest file

To enable the application cache for an app, include the manifest attribute on the document's `html` tag:

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

The `manifest` attribute should be included on every page of your web application that you want cached. The browser does not cache a page if it does not contain the `manifest` attribute (unless it is explicitly listed in the manifest file itself. This means that any page the user navigates to that includes a `manifest` will be implicitly added to the application cache. Thus, there's no need to list every page in your manifest. If a page points to a manifest, there's no way to prevent this page being cached.

"/page-url/", "/page-url/?something", "/page-url/?something-else" are considered separate pages. If they link to the manifest, they will all be implicitly cached separately. Because of this and other gotchas, AppCache best used on apps with one URL.

You can see the urls that are controlled by the application cache by visiting about:///appcache-internals/ in Chrome. From here you can clear caches and view the entries. There are similar developer tools in Firefox.

The `manifest` attribute can point to an absolute URL or relative path, but an absolute URL must be under the same origin as the web application. A manifest file can have any file extension, but needs to be served with the correct mime-type (see below).

```
<html manifest="http://www.example.com/example.mf">
  ...
</html>
```

A manifest file must be served with the mime-type `text/cache-manifest`. You may need to add a custom file type to your web server or `.htaccess` configuration.

For example, to serve this mime-type in Apache, add this line to your config file:

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

Or, in your app.yaml file in Google App Engine:

```
- url: /mystaticdir/(.*\.appcache)
  static_files: mystaticdir/\1
  mime_type: text/cache-manifest
  upload: mystaticdir/(.*\.appcache)
```

This requirement was dropped from the specification some time ago, and no longer required by the latest versions of Chrome, Safari and Firefox, but you'll need the mime-type to work in older browsers and IE11.

Structure of a manifest file

The manifest is a separate file you link to via the manifest attribute on the html element. A simple manifest looks something like this:

```
CACHE MANIFEST
index.html
stylesheet.css
images/logo.png
scripts/main.js
http://cdn.example.com/scripts/main.js
```

This example will cache four files on the page that specifies this manifest file.

There are a couple of things to note:

- The `CACHE MANIFEST` string is the first line and is required.
- Files can be from another domain
- Some browsers place restrictions on the amount of storage quota available to your app. In Chrome for example, AppCache uses a shared pool of TEMPORARY storage that other offline APIs can share. If you are writing an app for the Chrome Web Store, using the `unlimitedStorage` removes that restriction.
- If the manifest itself returns a 404 or 410, the cache is deleted.
- If the manifest or a resource specified in it fails to download, the entire cache update process fails. The browser will keep using the old application cache in the event of failure.

Lets take a look at a more complex example:

```
CACHE MANIFEST
# 2010-06-18:v2

# Explicitly cached 'master entries'.
```

```
CACHE:
/favicon.ico
index.html
stylesheet.css
images/logo.png
scripts/main.js

# Resources that require the user to be online.
NETWORK:
*

# static.html will be served if main.py is inaccessible
# offline.jpg will be served in place of all images in images/large/
# offline.html will be served in place of all other .html files
FALLBACK:
/main.py /static.html
images/large/ images/offline.jpg
```

Lines starting with a '#' are comment lines, but can also serve another purpose. An application's cache is only updated when its manifest file changes. So for example, if you edit an image resource or change a JavaScript function, those changes will not be re-cached. **You must modify the manifest file itself to inform the browser to refresh cached files**.

Avoid using a continually-updating timestamp or random string to force updates every time. The manifest is checked twice during an update, once at the start and once after all cached files have been updated. If the manifest has changed during the update, it's possible the browser fetched some files from one version, and other files from another version, so it doesn't apply the cache and retries later.

Although the cache updates, the browser won't use those files until the page is refreshed, because updates happen after the page is loaded from the current version of the cache.

A manifest can have three distinct sections: `CACHE`, `NETWORK`, and `FALLBACK`.

`CACHE:` : This is the default section for entries. Files listed under this header (or immediately after the `CACHE MANIFEST`) will be explicitly cached after they're downloaded for the first time. `NETWORK:` : Files listed in this section may come from the network if they aren't in the cache, otherwise the network isn't used, even if the user is online. You can white-list specific URLs here, or simply "", *which allows all URLs. Most sites need* "". `FALLBACK:` : An optional section specifying fallback pages if a resource is inaccessible. The first URI is the resource, the second is the fallback used if the network request fails or errors. Both URIs must from the same origin as the manifest file. You can capture specific URLs but also URL prefixes. "images/large/" will capture failures from URLs such as "images/large/whatever/img.jpg".

These sections can be listed in any order and each section can appear more than one in a single manifest.

The following manifest defines a "catch-all" page (offline.html) that will be displayed when the user tries to access the root of the site while offline. It also declares that all other resources (e.g. those on remote a site) require an internet connection.

```
CACHE MANIFEST
# 2010-06-18:v3

# Explicitly cached entries
index.html
css/style.css

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

# All other resources (e.g. sites) require the user to be online.
NETWORK:
*

# Additional resources to cache
CACHE:
images/logo1.png
images/logo2.png
images/logo3.png
```

The HTML file that references your manifest file is automatically cached. There's no need to include it in your manifest, however it is encouraged to do so.

HTTP cache headers and the caching restrictions imposed on pages served over SSL are overridden by cache manifests. Thus, pages served over https can be made to work offline.

Updating the cache

Once an application is offline it remains cached until one of the following happens:

1. The user clears their browser's data storage for your site.
2. The manifest file is modified. Note: updating a file listed in the manifest doesn't mean the browser will re-cache that resource. The manifest file itself must be altered.

Cache status

The `window.applicationCache` object is your programmatic access the browser's app cache. Its `status` property is useful for checking the current state of the cache:

```
var appCache = window.applicationCache;


switch (appCache.status) {
case appCache.UNCACHED: // UNCACHED == 0
return 'UNCACHED';
break;
case appCache.IDLE: // IDLE == 1
return 'IDLE';
break;
case appCache.CHECKING: // CHECKING == 2
return 'CHECKING';
break;
case appCache.DOWNLOADING: // DOWNLOADING == 3
return 'DOWNLOADING';
break;
case appCache.UPDATEREADY:  // UPDATEREADY == 4
return 'UPDATEREADY';
break;
case appCache.OBSOLETE: // OBSOLETE == 5
return 'OBSOLETE';
break;
default:
return 'UKNOWN CACHE STATUS';
break;
};
```

To programmatically check for updates to the manifest, first call `applicationCache.update()`. This will attempt to update the user's cache (which requires the manifest file to have changed). Finally, when the `applicationCache.status` is in its `UPDATEREADY` state, calling `applicationCache.swapCache()` will swap the old cache for the new one.

```
var appCache = window.applicationCache;


appCache.update(); // Attempt to update the user's cache.
```

```
...

if (appCache.status == window.applicationCache.UPDATEREADY) {
appCache.swapCache();  // The fetch was successful, swap in the new cache.
}
```

Using `update()` and `swapCache()` like this will mean the new cache will be used for subsequent downloads, but the user will already have downloaded the page and likely all resources by this point, those files won't be automatically reloaded. You'll need to refresh the page to get the latest version of the page and resources, which doesn't require calling `swapCache()`

The good news: you can automate this. To update users to the newest version of your site, set a listener to monitor the `updateready` event on page load:

```
// Check if a new cache is available on page load.
window.addEventListener('load', function(e) {

window.applicationCache.addEventListener('updateready', function(e) {
if (window.applicationCache.status == window.applicationCache.UPDATEREADY) {
    // Browser downloaded a new app cache.
    if (confirm('A new version of this site is available. Load it?')) {
    window.location.reload();
    }
} else {
    // Manifest didn't changed. Nothing new to server.
}
}, false);

}, false);
```

AppCache events

As you may expect, additional events are exposed to monitor the cache's state. The browser fires events for things like download progress, updating the app cache, and error conditions. The following snippet sets up event listeners for each type of cache event:

```
function handleCacheEvent(e) {
//...
}
```

```
function handleCacheError(e) {
alert('Error: Cache failed to update!');
};

// Fired after the first cache of the manifest.
appCache.addEventListener('cached', handleCacheEvent, false);

// Checking for an update. Always the first event fired in the sequence.
appCache.addEventListener('checking', handleCacheEvent, false);

// An update was found. The browser is fetching resources.
appCache.addEventListener('downloading', handleCacheEvent, false);

// The manifest returns 404 or 410, the download failed,
// or the manifest changed while the download was in progress.
appCache.addEventListener('error', handleCacheError, false);

// Fired after the first download of the manifest.
appCache.addEventListener('noupdate', handleCacheEvent, false);

// Fired if the manifest file returns a 404 or 410.
// This results in the application cache being deleted.
appCache.addEventListener('obsolete', handleCacheEvent, false);

// Fired for each resource listed in the manifest as it is being fetched.
appCache.addEventListener('progress', handleCacheEvent, false);

// Fired when the manifest resources have been newly redownloaded.
appCache.addEventListener('updateready', handleCacheEvent, false);
```

If the manifest file or a resource specified in it fails to download, the entire update fails. The browser will continue to use the old application cache in the event of such a failure.

References

- ApplicationCache API specification
- Application Cache is a douchebag - covering the gotchas and issues with AppCache.

□□□ #3
□ □□□ □□□ 2026-10-31 21:36:59 UTC

 cocobook-sync □□□ 2026-11-04 17:36:31 UTC