A Beginner's Guide to Using the Application Cache

Eric Bidelman - Developer Relations, Google

June 18, 2010 (updated: May 27, 2011)

Supported browsers:

Your browser appears to support all of the functionality used in this article.

Introduction

It's becoming increasingly important for web-based applications to be accessible offline. Yes, all browsers have caching mechanisms, but they're unreliable and don't always work as you might expect. 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 - cached resources are local, and therefore load faster.
3. Reduced server load - the browser will only download resources from the server that have changed.

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 include a `manifest` will be implicitly added to the application cache. Thus, there's no need to list every page in your manifest.

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)
```

Structure of a manifest file

A simple manifest may look something like this:

```
CACHE MANIFEST
index.html
stylesheet.css
images/logo.png
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.
- Sites are limited to 5MB worth of cached data. However, if you are writing an app for the Chrome Web Store , using the `unlimitedStorage` removes that restriction.
- If the manifest file 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:
login.php
/myapi
http://api.twitter.com

# 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
*.html /offline.html
```

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**. Creating a comment line with a generated version number, hash of your files, or timestamp is one way to ensure users have the latest version of your software. You can also programmatically update the cache once a new version is ready as discussed in the Updating the cache section.

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 under this section are white-listed resources that require a connection to the server. All requests to these resources bypass the cache, even if the user is offline. Wildcards may be used.

`FALLBACK:`
: An optional section specifying fallback pages if a resource is inaccessible. The first URI is the resource, the second is the fallback. Both URIs must be relative and from the same origin as the manifest file. Wildcards may be used.

Note: 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
```

Note: 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.

Note: 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 alternated.
3. The app cache is programatically updated.

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 update the cache, 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.
}
```

Note: Using `update()` and `swapCache()` like this does not serve the updated resources to users. This flow simply tells the browser to check for a new manifest, download the updated content it specifies, and repopulate the app cache. Thus, it takes two page reloads to server new content to users, one to pull down a new app cache, and another to refresh the page content.

The good news: you can avoid this double reload headache. 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.
      // Swap it in and reload the page to get the new hotness.
      window.applicationCache.swapCache();
      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

Showing 14 comments Sort by popular now ▾

Nicholas Cardot

What can be done to cache everything except the page that is calling the file. I have a dynamic app, I want to cache all the supporting CSS, images, etc, but I don't want the actual pages to be cached. Will listing the pages in the NETWORK section be enough to over ride the automatic inclusion of files that call the cache?

3 months ago 1 Like Like Reply

Viktor

you can workaround this with an iframe, http://saikotroid.blogspot.com...

2 months ago in reply to Nicholas Cardot Like Reply

solsTiCe d'Hiver

Is there anything possible for conditional caching ?

[EDIT]
I have removed my question because IE does not support html5 caching

3 weeks ago Like Reply

Rich

for a large website, can the web server automatically update and version the manifest file if it detects changes to files, or should this be handled by a site admin/script?

1 month ago Like Reply

Eric Bidelman

Sure. Check this dymanic manifest out: https://github.com/ebidel/html...

It changes the "version comment string" only when one of the manifest's file content changes.
That's down by checking an md5 of all of the files' content.

1 month ago in reply to Rich Like Reply

Jaydeep

The AddType Configurations are to be done in the httpd.conf file if you are using Apache Web Server.

1 month ago Like Reply

Jaydeep

Caching the page which contains the manifest.appcache file is not recommended. Keeping such a page in NETWORK: part of manifest.appcache helps in updation of the Applicaiton Cache everytime the website is browsed.

1 month ago Like Reply

Eric Bidelman

Putting the master entry in the NETWORK: section won't work. The page that references the mainfest file (the master entry) is always cached. There's no way around that.

1 month ago in reply to Jaydeep Like Reply

Brian

I'm a little late to this party, but hopefully someone here can help. In Safari/Chrome the appcache works for me as expected. In FF (5.0.1) the cache re-downloads on every page visit...as though the manifest has been changed. Anyone else see this or have any insight?

2 months ago Like Reply

DCam

UPDATEREADY == 5
should beUPDATEREADY == 4

4 months ago Like Reply

Eric Bidelman

Good catch. I'll make the fix. Thanks!

4 months ago in reply to DCam Like Reply

MacHarborGuy

Here is a tip when building an HTML5 web app using Cache Manifest. If you are using PHP or another dynamic scripting language is to use a GET variable to filter out the Cache Manifest.

I am currently building a web app and want to test it out in both a cached and non-cached state. Instead of having two separate instances, I use a GET variable to filter out the

Cache attribute.

On my iPhone I have two Home Screen bookmarks, one leading to the URL and another with a GET variable with some hard to guess string. If the GET variable exists, the cache attribute is never added to the document, and never forces the browser to cache the site. If it does not exist, the site is cached.

The nice part about the way the iPhone handles Home Screen WebApp bookmarks is that...

http://www.example.com

and...

http://www.example.com/?var=1

are treated as separate instances when the Cache is concerned.

This allows me to, as long as I don't modify the cache.manifest file, have a live cached version of the app for testing, and a non-cached version to quickly preview changes without having to even touch the manifest (which can be very obtrusive during the initial development).

What is even better is the ability to whitelist remote-server URLs (the Google domains, for example, with the Google Maps Javascript API) in the NETWORK: section. Just make sure you get all of the remote URLs whitelisted, otherwise you will have an app that doesn't work. So far I have 6 Google URLs whitelisted. I am guessing there are more, but at this point I have not run into them.

6 months ago Like Reply

Louis-Rémi Babé

"These entries (i.e. everything under a CACHE heading) must follow the same origin policy"
Are you sure? Caching a script from a different domain seems to work for me...

6 months ago Like Reply

Eric Bidelman

You're right Louis. The CACHE entries can be from a different origin. It's the manifest attribute on that must be from the same origin. I'll update those sections. Thanks!

6 months ago in reply to Louis-Rémi Babé 3 Likes Like Reply