Application Cache chrome://appcache-internals/

## What is Application Cache?

HTML5 introduces application cache, which means that a web application is cached, and accessible without an internet connection.

Application cache gives an application three advantages:

1. Offline browsing - users can use the application when they're offline
2. Speed - cached resources load faster
3. Reduced server load - the browser will only download updated/changed resources from the server

## HTML Cache Manifest Example

The example below shows an HTML document with a cache manifest (for offline browsing):

<!DOCTYPE HTML>  
<html manifest="demo.appcache">  
  
<body>  
The content of the document......  
</body>  
  
</html>

Every page with the manifest attribute specified will be cached when the user visits it. If the manifest attribute is not specified, the page will not be cached (unless the page is specified directly in the manifest file).

The recommended file extension for manifest files is: ".appcache"

A manifest file needs to be served with the **correct media type**, which is "text/cache-manifest". Must be configured on the web server.

## The Manifest File

The manifest file is a simple text file, which tells the browser what to cache (and what to never cache).

The manifest file has three sections:

* **CACHE MANIFEST** - Files listed under this header will be cached after they are downloaded for the first time
* **NETWORK** - Files listed under this header require a connection to the server, and will never be cached
* **FALLBACK** - Files listed under this header specifies fallback pages if a page is inaccessible

### CACHE MANIFEST

The first line, CACHE MANIFEST, is required:

CACHE MANIFEST  
/theme.css  
/logo.gif  
/main.js

The manifest file above lists three resources: a CSS file, a GIF image, and a JavaScript file. When the manifest file is loaded, the browser will download the three files from the root directory of the web site. Then, whenever the user is not connected to the internet, the resources will still be available.

### NETWORK

The NETWORK section below specifies that the file "login.asp" should never be cached, and will not be available offline:

NETWORK:  
login.php

An asterisk can be used to indicate that all other resources/files require an internet connection:

NETWORK:  
\*

### FALLBACK

The FALLBACK section below specifies that "offline.html" will be served in place of all files in the /html/ catalog, in case an internet connection cannot be established:

FALLBACK:  
/html/ /offline.html

**Note:** The first URI is the resource, the second is the fallback.

## Updating the Cache

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

* The user clears the browser's cache
* The manifest file is modified (see tip below)
* The application cache is programmatically updated

## Example - Complete Cache Manifest File

CACHE MANIFEST  
# 2012-02-21 v1.0.0  
/theme.css  
/logo.gif  
/main.js  
  
NETWORK:  
login.asp  
  
FALLBACK:  
/html/ /offline.html

html/ /offline.html

**Tip:** 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. If you edit an image or change a JavaScript function, those changes will not be re-cached. Updating the date and version in a comment line is one way to make the browser re-cache your files.

## Notes on Application Cache

Be careful with what you cache.

Once a file is cached, the browser will continue to show the cached version, even if you change the file on the server. To ensure the browser updates the cache, you need to change the manifest file.

**Note:** Browsers may have different size limits for cached data (some browsers have a 5MB limit per site).

Loading documents

The use of an application cache modifies the normal process of loading a document:

* If an application cache exists, the browser loads the document and its associated resources directly from the cache, without accessing the network. This speeds up the document load time.
* The browser then checks to see if the cache manifest has been updated on the server.
* If the cache manifest has been updated, the browser downloads a new version of the manifest and the resources listed in the manifest. This is done in the background and does not affect performance significantly

## Resources in an application cache[EDIT](https://developer.mozilla.org/en-US/docs/Web/HTML/Using_the_application_cache$edit#Resources_in_an_application_cache)[EDIT](file:///C:\en-US\docs\Web\HTML\Using_the_application_cache$edit#Resources_in_an_application_cache)

An application cache always includes at least one resource, identified by URI. All resources fit into one of the following categories:

**Master entries**

These are resources added to the cache because a browsing context visited by the user included a document that indicated that it was in this cache using its manifest attribute.

**Explicit entries**

These are resources explicitly listed in the application's cache manifest file.

**Network entries**

These are resources listed in the application's cache manifest files as network entries.

**Fallback entries**

These are resources listed in the application's cache manifest files as fallback entries.

**Note:** Resources can be tagged with multiple categories, and can therefore be categorized as multiple entries. For example, an entry can be both an explicit entry and a fallback entry.

### Master entries

Master entries are any HTML files that include a [manifest](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/html#attr-manifest) attribute in their [<html>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/html) element. For example, let's say we have the HTML file <http://www.example.com/entry.html>, which looks like this:

<html manifest="example.appcache">

<h1>Application Cache Example</h1>

</html>

If entry.html is not listed in the example.appcache cache manifest file, visiting the entry.htmlpage causes entry.html to be added to the application cache as a master entry.

### Explicit entries

Explicit entries are resources that are explicitly listed in the CACHE section of a cache manifest file.

### Network entries

The NETWORK section of a cache manifest file specifies resources for which a web application requires online access. Network entries in an application cache are essentially an "online whitelist"—URIs specified in the NETWORK section are loaded from the server instead of the cache. This lets the browser's security model protect the user from potential security breaches by limiting access to approved resources.

As an example, you can use network entries to load and execute scripts and other code from the server instead of the cache:

CACHE MANIFEST

NETWORK:

/api

The cache manifest section listed above ensures that requests to load resources contained in the <http://www.example.com/api/> subtree always go to the network without attempting to access the cache.

**Note**: Simply omitting master entries (files that have the manifest attribute set in the html element) from the manifest file would not have the same result, because master entries will be added—and subsequently served from—the application cache.

### Fallback entries

Fallback entries are used when an attempt to load a resource fails. For example, let's say the cache manifest file <http://www.example.com/example.appcache> includes the following content:

CACHE MANIFEST

FALLBACK:

example/bar/ example.html

Any request to <http://www.example.com/example/bar/> or any of its subdirectories and their content cause the browser to issue a network request to attempt to load the requested resource. If the attempt fails, due to either a network failure or a server error of some kind, the browser loads the file example.html instead.

## Cache states[EDIT](https://developer.mozilla.org/en-US/docs/Web/HTML/Using_the_application_cache$edit#Cache_states)[EDIT](file:///C:\en-US\docs\Web\HTML\Using_the_application_cache$edit#Cache_states)

Each application cache has a **state**, which indicates the current condition of the cache. Caches that share the same manifest URI share the same cache state, which can be one of the following:

**UNCACHED**

A special value that indicates that an application cache object is not fully initialized.

**IDLE**

The application cache is not currently in the process of being updated.

**CHECKING**

The manifest is being fetched and checked for updates.

**DOWNLOADING**

Resources are being downloaded to be added to the cache, due to a changed resource manifest.

**UPDATEREADY**

There is a new version of the application cache available. There is a corresponding updateready event, which is fired instead of the cached event when a new update has been downloaded but not yet activated using the swapCache() method.

**OBSOLETE**

The application cache group is now obsolete.

## Testing for updates to the cache manifest

You can programmatically test to see if an application has an updated cache manifest file, using JavaScript. Since a cache manifest file may have been updated before a script attaches event listeners to test for updates, scripts should always test window.applicationCache.status.

function onUpdateReady() {

console.log('found new version!');

}

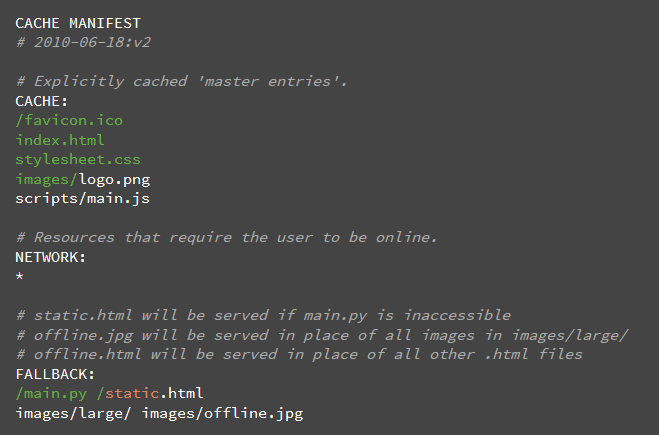
window.applicationCache.addEventListener('updateready', onUpdateReady);

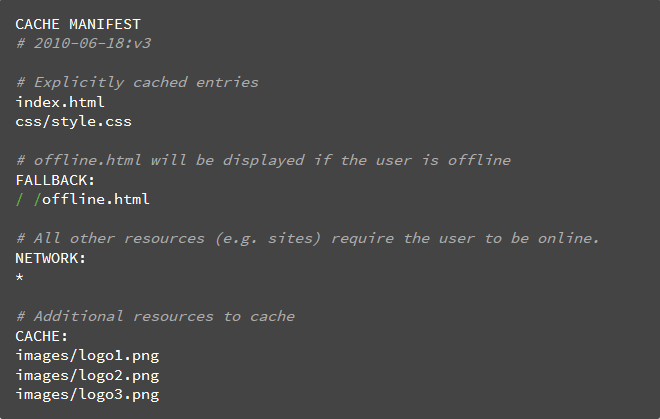
if(window.applicationCache.status === window.applicationCache.UPDATEREADY) {

onUpdateReady();

}

To manually start testing for a new manifest file, you can use window.applicationCache.update().





## 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.

}

**Note**: 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.