Background Fetch Design Document

Jennifer Harkness, Jake Archibald, Peter Beverloo, Anita Woodruff
January 2017

[References](#)

[Objective](#)

[Staged Delivery of BackgroundFetch](#)

[Overview of the Design](#)

[Details of the Design](#)
- [Mojo Service](#)
- [BackgroundFetchManager](#)
- [BackgroundFetchDataManager](#)
- [BackgroundFetchDataStorage](#)
- [BackgroundFetchBundleManager](#)
- [BackgroundFetchDownloadObserver](#)

[UX](#)

[Security Considerations](#)

[Privacy Considerations](#)

[Resource Consumption](#)
- [Battery & RAM](#)
- [Data](#)
- [Disk](#)

[Testing Plan](#)

[Metrics](#)

References

- [Github](#) proposal
- Partial [specification](#)
- [Master bug](#)

Objective

Service workers are capable of fetching and caching assets, the size of which is unrestricted. However, if the user navigates away from the site or closes the browser, the service worker is

likely to be killed. This can happen even if there's a pending promise passed to extendableEvent.waitUntil, if it hasn't resolved within a few minutes the browser may consider it an abuse of service workers and kill the worker.

This makes it difficult to download or upload large assets such as podcasts, movies, or photos. Even if the service worker isn't killed, having to keep the service worker and therefore the browser in memory during this potentially long operation is wasteful.

Long term goals:

- Enable background-caching of multiple resources, both triggered from a foreground tab or from a service worker.
- Enable background-uploading of multiple resources, both triggered from a foreground tab or from a service worker.
- Show UI to indicate the progress of the fetch, and allow the user to pause/abort.
- Allow the OS to handle the fetch, so the browser doesn't need to continue running.
- Allow the OS to deal with poor connectivity by pausing/resuming the download/upload.
- Allow the app to react to success/failure of the fetch, perhaps by showing a notification

Staged Delivery of BackgroundFetch

As is clear from the Objectives, there are a large number of features involved in BackgroundFetch, more than are reasonable to implement in a single pass. As a result, the team has decided to separate the feature into stages.

The initial implementation will have the following limitations:

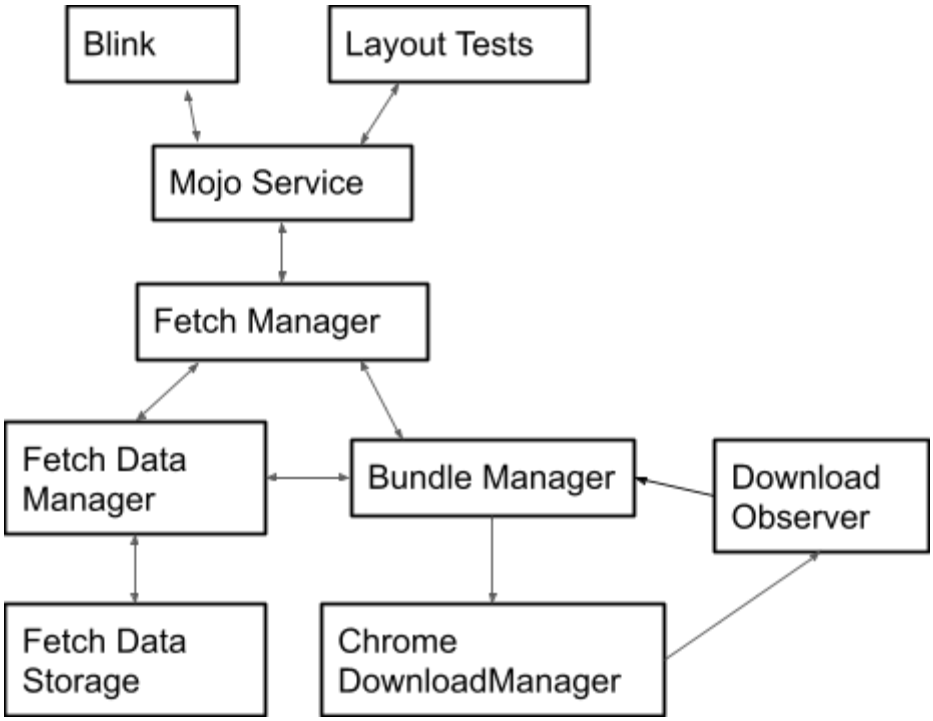
- GET only. Uploads can be implemented later.
- Downloads will be handled by the Chrome Download Manager, so the foreground notification will be shown by the DownloadManager.
- Single files only. The API will take a sequence of Requests, but the implementation will require that the number of files in the list is 1.
- No complex permission systems. The visibility of the download provides user control.
- Initial launch through the experimental framework, so developer use can inform the final API.

There are a large number of complexities in the UX, API, and privacy areas that these limitations significantly reduce. This document will mention those complexities throughout, but only has detailed solutions for the subset which are needed for the initial implementation.

Overview of the Design

At a very high level, the BackgroundFetch system will be a dispatcher, which takes requests from developer code and dispatches them to the Download system. It keeps enough local data that when the result is available from the download system, it can dispatch the response back to the developer code, even if that code is no longer loaded.

Details of the Design



Mojo Service

We'll have BackgroundFetchService which will be a Mojo service. The BackgroundFetchServiceImpl will live in content and will provide an implementation which talks to the BackgroundFetchManager.

BackgroundFetchManager

The BackgroundFetchManager will live in content/browser/background_fetch and will be the coordinator of all background fetch operations. It will be responsible for dispatching requests to the Chrome download system, and will be the component that is notified when a download

completes. It will be responsible for reconnecting to the download system after Chrome restarts and checking on the status of ongoing downloads.

The BackgroundFetchManager will be a KeyedService and will have its lifetime tied to the lifetime of a user Profile.

BackgroundFetchDataManager

The BackgroundFetchDataManager is responsible for maintaining all the metadata about outstanding requests. This includes information about which service worker or document should be notified when a request completes and details of which individual files are grouped together in a named request. The BackgroundFetchDataManager also must write this information to some form of stable storage so that it is available after browser restarts.

Open question: What do we do if a service worker version updates? Invalidate existing requests? Continue them?

BackgroundFetchDataStorage

Wrapper class around a LevelDB ProtoDatabase which will take care of storage for the DataManager. Each DataManager will have a DataStorage. It will support request lookup by ID and request lookup by service worker.

BackgroundFetchBundleManager

The BackgroundFetchBundleManager is responsible for keeping track of multiple files that are bundled together by the caller into a single request. This component monitors status of each of the individual downloads via the BackgroundFetchDownloadObserver and updates the FetchManager when all of the individual downloads have resolved.

BackgroundFetchDownloadObserver

Implements DownloadManager::Observer and reports back to the BackgroundFetchBundleManager on updates to the files being tracked by the background fetch system.

UX

For the initial implementation, the UX will be entirely dependent on the UI of the existing Download Manager, since the Background Fetch system will pass requests there. Depending on the recommendations of UX designers, we may try to add features to the Download Manager for the initial release such as:

- Grouping multiple downloads under a single visible name.
- Attributing downloads to particular origins.
- Marking downloads as background initiated.
- Branding downloads with origin icons.

For the long-term plan, we are working with other groups who are trying to solve the UX problem of showing offline content and the UX team is actively involved in those discussions. The end result will likely be an API which the BackgroundFetch system will invoke, but will not have a large impact on the design of the internals of the system itself.

Security Considerations

Websites can already exercise downloads without user gesture. We will need to be very careful on the implementation of uploads to avoid security issues such as uploading data that the user expects to remain on their local device.

The DownloadManager has different download paths for downloading with or without a frame. Downloads with a frame are handled by the ResourceDispatcherHostImpl, which hands off to the ResourceLoader after doing checks such as whether the frame's security policy allows access of the given URL. However, BackgroundFetch will be using downloads without a frame, which are currently processed by a URLDownloader, which doesn't check things like a security policy. The BackgroundFetch system will need to do those checks before issuing the download instead.

We will be doing a full security review as part of this project, so some security issues will be addressed then.

Privacy Considerations

Requests can outlive the tab, hence may be sent on different networks to the network the page was requested on or a push message was received on. However, as long as the fetch is ongoing, there will be a download notification, so the user will be aware that data is being transmitted. The user has the option to pause the download if they do not want the download going across a particular network.

There is also a subtlety with cookies affected one-shot background sync and will also affect background fetch.

There is a content setting for one-shot background sync (“Background sync - Allow recently closed sites to finish sending and receiving data”; see wording discussions) that lets users globally disable that feature. We may want to broaden the wording of that content setting, or add a separate one. This will be discussed with the UX and privacy teams.

There will be a full privacy review as part of this project.

Resource Consumption

Battery & RAM

Chrome’s usual Downloads system will be used.

Note that we wouldn’t be able to use Android’s DownloadManager since it doesn’t support POST requests. It may be possible to extend DownloadManager in future versions of Android, however it takes a long time for such platform changes to reach a large percentage of devices; a more promising avenue might be to add a download manager to Google Play Services (which does reach older devices) if we think it would be of use to other Android apps.

Long-term: consider allowing websites to choose whether to wait until the device is charging?

Data

Short-term: rely on Chrome Downloads for behavior.
Long-term: consider allowing websites to choose whether to pause when exiting WiFi.

Disk

Fetch responses will be saved to files under Chrome’s app data directory.

- Should we mark them as cached data (hence easier for OS to discard)?
- Need to ensure doesn’t exceed storage quota, or at least not for very long (especially with multiple simultaneous downloads). This is tricky, since we may not know response size until it finishes downloading.

Testing Plan

- Unit tests: Each of the individual components in content/chrome/background_fetch will have a unit test.

- Layout tests: There will be a full interface test, and also a functionality test which mocks the fetch manager with a JavaScript implementation to test the Mojo service.
- Browser tests: There will be an end-to-end browser test to validate download completion and notification of the service worker, including tests where the service worker is in the background.

Metrics

- API metrics - we'll have UMA for each of the API calls to evaluate developer use.
- Bundle metrics - we'll have one UMA for requested bundle size and another API for bundle success rate.
- Overall download time - A metric will measure how long in active network time background fetch downloads take to complete.
- Overall download size - A metric to observe the average size of downloads using background fetch.