

Introducing Background Sync



By Jake Archibald

Human boy working on web standards at Google

Background sync is a new web API that lets you defer actions until the user has stable connectivity. This is useful for ensuring that whatever the user wants to send, is actually sent.

The problem

The internet is a great place to waste time. Without wasting time on the internet, we wouldn't know cats dislike flowers, chameleons love bubbles, or that our very own Eric Bidelman is a putt putt golfing hero of the late 90s.

But sometimes, just sometimes, we're not looking to waste time. The desired user experience is more like:

1. Phone out of pocket.
2. Achieve minor goal.
3. Phone back in pocket.
4. Resume life.

Unfortunately this experience is frequently broken by poor connectivity. We've all been there. You're staring at a white screen or a spinner, and you know you should just give up and get on with your life, but you give it another 10 seconds just in case. After that 10 seconds? Nothing. But why give up now? You've invested time already, so walking away with nothing would be a waste, so you carry on waiting. By this point you *want* to give up, but you know the second you do so, is the second before everything would have loaded if only you'd waited.

Service workers solve the page loading part by letting you serve content from a cache. But what about when the page needs to send something to the server?

At the moment, if the user hits "send" on a message they have to stare at a spinner until it completes. If they try to navigate away or close the tab, we use onbeforeunload to display a

message like, “Nope, I need you to stare at this spinner some more. Sorry”. If the user has no connection we tell the user “Sorry, *you* must come back later and try again”.

This is rubbish. Background sync lets you do better.

The solution

The following video shows Emojoy, a simple emoji-only chat demo... thing. It's a progressive web app. It works offline-first. It uses push messages and notifications, and it uses background sync.


If the user tries to send a message when they have zero connectivity, then, thankfully, the message is sent in the background once they get connectivity.

As of March 2016, Background sync is available in Chrome from version 49 and above. You can see it action by following the steps below:

1. Open Emojoy.
2. Go offline (either using airplane-mode or visit your local Faraday cage).
3. Type a message.
4. Go back to your home screen (optionally close the tab/browser).
5. Go online.
6. Message sends in the background!

Being able to send in the background like this also yields a perceived performance improvement. The app doesn't need to make such a big deal about the message sending, so it can add the message to the output straight away.

How to request a background sync

In true [extensible web](#)  style, this is a low level feature that gives you the freedom to do what you need. You ask for an event to be fired when the user has connectivity, which is immediate if the user already has connectivity. Then, you listen for that event and do whatever you need to do.

Like push messaging, it uses a [service worker](#) as the event target, which enables it to work when the page isn't open. To begin, register for a sync from a page:

```
// Register your service worker:
navigator.serviceWorker.register('/sw.js');

// Then later, request a one-off sync:
navigator.serviceWorker.ready.then(function(swRegistration) {
  return swRegistration.sync.register('myFirstSync');
});
```




Then listen for the event in `/sw.js`:

```
self.addEventListener('sync', function(event) {
  if (event.tag == 'myFirstSync') {
    event.waitUntil(doSomeStuff());
  }
});
```



And that's it! In the above, `doSomeStuff()` should return a promise indicating the success/failure of whatever it's trying to do. If it fulfills, the sync is complete. If it fails, another sync will be scheduled to retry. Retry syncs also wait for connectivity, and employ an exponential back-off.

The tag name of the sync ('myFirstSync' in the above example) should be unique for a given sync. If you register for a sync using the same tag as a pending sync, it coalesces with the existing sync. That means you can register for an "clear-outbox" sync every time the user sends a message, but if they send 5 messages while offline, you'll only get one sync when they become online. If you want 5 separate sync events, just use unique tags!

[Here's a simple demo](#)  that does the bare minimum; it uses the sync event to show a notification.

What could I use background sync for?

Ideally, you'd use it to schedule any data sending that you care about beyond the life of the page. Chat messages, emails, document updates, settings changes, photo uploads...

anything that you want to reach the server even if user navigates away or closes the tab. The page could store these in an "outbox" store in indexedDB, and the service worker would retrieve them, and send them.

Although, you could also use it to fetch small bits of data...

Another demo!

This is the offline wikipedia demo I created for Supercharging Page Load. I've since added some background sync magic to it.

Try this out yourself. Make sure you are using Chrome 49 and above and then:

1. Go to any article, perhaps Chrome.
2. Go offline (either using airplane-mode or join a terrible mobile provider like I have).
3. Click a link to another article.
4. You should be told the page failed to load (this will also appear if the page just takes a while to load).
5. Agree to notifications.
6. Close the browser.
7. Go online
8. You get notified when the article is downloaded, cached, and ready to view!

Using this pattern, the user can put their phone in their pocket and get on with their life, knowing the phone will alert them when it's fetched what they wanted.

Permissions

The demos I've shown use web notifications [↗](#), which require permission, but background sync itself does not.

Sync events will often complete while the user has a page open to the site, so requiring user permission would be a poor experience. Instead, we're limiting when syncs can be registered and triggered to prevent abuse. E.g.:

- You can only register for a sync event when the user has a window open to the site.
- The event execution time is capped, so you can't use them to ping a server every x seconds, mine bitcoins or whatever.

Of course, these restrictions may loosen/tighten based on real-world usage.

Progressive enhancement

It'll be a while before all browsers support background sync, especially as Safari and Edge don't yet support service workers. But progressive enhancement helps here:

```
if ('serviceWorker' in navigator && 'SyncManager' in window) {  
  navigator.serviceWorker.ready.then(function(reg) {  
    return reg.sync.register('tag-name');  
  }).catch(function() {  
    // system was unable to register for a sync,  
    // this could be an OS-level restriction  
    postDataFromThePage();  
  });  
} else {  
  // serviceworker/sync not supported  
  postDataFromThePage();  
}
```



If service workers or background sync aren't available, just post the content from the page as you'd do today.

Note that it's worth using background sync even if the user appears to have good connectivity, as it protects you against navigations and tab closures during data send.

The future

We're aiming to ship background sync to a stable version of Chrome in the first half of 2016. But we're also working on a variant, "periodic background sync". This will allow you to request a "periodicsync" event restricted by time interval, battery state and network state. This would require user permission, of course, but it will also be down to the will of the

browser for when and how often these events fire. E.g., a news site could request to sync every hour, but the browser may know you only read that site at 07:00, so the sync would fire daily at 06:50. This idea is a little further off than one-off syncing, but it's coming.

Bit by bit we're bringing successful patterns from Android/iOS onto the web, while still retaining what makes the web great!

Except as otherwise noted, the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#). For details, see our [Site Policies](#). Java is a registered trademark of Oracle and/or its affiliates.

Last updated July 2, 2018.