Periodic Background Sync PRD

nator@
Contributors: peter@, rayankans@, jakearchibald@
Last modified: 01-15-2019
Status: [Draft | Under Review | Reviewed | Implemented | Obsolete]
This document is public.

Problem statement/Situational Overview

For motivation and partner requests, see this doc.
The feature will enable web apps to register tasks to be run periodically, when there's network connectivity, to allow apps to update state and download/upload content.

Target user

- Web developers - who'll be able to provide a richer web experience using this API.

Goals

- Enable a web app to register tasks to be run periodically, with a retry logic baked in, which has an exponential backoff.
- Provide the web app a way to query the list of previously registered tasks.
- Enable the web app to unregister previously registered tasks, or provide an expire_after at the time of registration.
- Supported platforms: Android, Windows, Linux, ChromeOS, Mac.

Non-goals

- Triggering events at a specific time is an explicit non-goal. The Scheduled tasks API enables that.
- Notifying the user for each periodic Sync
- Unified System UI to show background activity of web apps at any given time.
- A unified setting that enables the user to allow Chrome to let web apps do tasks in the background, beyond their visible lifetime. This would also let the user control which type of network these tasks can run on.
- Unsupported platforms: iOS, WebView. WebView will support the onsync event so long as there is a controlled client but will not fire in the background. This way pages that use background sync can function normally in the foreground.
- Multiple periodic tasks per origin, with varying frequency.

Solution

Add periodicSync as a read-only property of ServiceWorkerRegistration, which would return a PeriodicSyncManager object.

PeriodicSyncManager will have a register, and an unregister, and a getTags() method on it.

```
  Promise<void> register(DOMString tag, optional
BackgroundSyncOptions options);
  Promise<void> unregister(DOMString tag);
```

ServiceWorkerGlobalScope would be extended to include an eventHandler called onPeriodicSync of PeriodicSyncEvent interface. The periodicSync event will provide the tag of the registration.

Example usage:
Register for a Periodic Sync from a page:

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

```
// Then later, request a periodic sync:
navigator.serviceWorker.ready.then(function(swRegistration) {
  return swRegistration.periodicSync.register(
      'daily_sync',  {minInterval: 200, delay: 24*60*60*1000});
});
```

Listen for event in the service worker:

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

Implementation design will be detailed in the design doc.

UX Walkthrough

No new UI or content setting will be added. There is however an existing content setting for Background Sync which will apply to both one-shot and periodic sync.

Privacy concerns

One-shot Background Sync allows the web page's logic to live a little longer (3 min*max 3 attempts) after the page has been closed by the user. Periodic Background Sync extends it to potentially infinite time thereafter. Here are some of the privacy concerns:

1. There is no notification to the user.
2. A periodic sync can be enabled while the user is connected to one network, and the sync event can be fired later when they're connected to another network. This can cause inadvertent leakage of browsing history on an unintended network. This concern applies to other background tasks such as one-shot Sync and Background Fetch as well.
3. Location tracking: The user's IP address can be revealed to the website every time the periodic task is run. This risk is also present with any tasks run in a service worker, but Periodic Sync allows persistent tracking. Note that this risk is also present with Push messages.

Mitigations of Privacy concerns

1. Chrome will limit the capability to installed web apps.
2. Chrome will limit the number of times Chrome is woken up each day to send Periodic Sync events. This will be set to twice a day, if there are pending Periodic Sync

registrations. A website can, thus, not run a task in the background more frequently than twice a day.
3. Chrome will use Site engagement to periodically prune the list of registrations for which we'll send sync events. In other words, we'll only send sync events for sites the user is actively engaged in.
4. Chrome will use Site engagement data to guide how often a site should be allowed to run code in the background, which will never be more often than twice a day.
5. Chrome will expose active registrations through dev tools UI.
6. The existing site setting to disable Background Sync will also apply to periodic Background Sync.

Resource concerns and mitigations

Unlike one-shot Sync, Periodic Syncs can cause a phone to wake up continually to run the periodic task. This will have a greater impact on battery and data usage than one-shot Sync, which wouldn't live too long beyond the lifetime of the page.

To alleviate this, we'll only wake the browser up twice a day and batch process any pending periodic sync registrations.

Further, we plan to carefully track resource usage, and if required, can implement mitigations like:

1. Reduce the time the periodic task is allowed to run for, especially for sites with a lower site engagement score.
2. Only run periodic tasks when the phone is on wifi.
3. Only run periodic tasks when the phone is not in battery drain mode.
4. Do not wake up the device to run periodic tasks.

Requirements

Description	Priority
Support registration of periodic tasks	P0
Support querying of registered periodic tasks	P0
Support unregistration of periodic tasks	P0
Support periodic automatic suspension of tasks based on site-engagement data	P2
Support running tasks periodically, which, on Android also involves potentially relaunching the browser a certain number of times per day.	P0

Metrics

Key success metrics

- Usage metrics, to track whether feature is being used.
- Ratio of success to attempts, to track reliability, especially for tasks that succeed a certain percentage of times (to rule out bugs in the JavaScript logic)
- Number of tasks registered per origin.
- Success rates of tasks per origin.

New Measurements

- Usage metrics
- Error reports

Rollout plan

- Origin Trial experiment for Periodic Background Sync, followed by,
- Launch, provided there's enough interest.

Alternatives considered

1. Other options considered, with a list of pros and cons.
2. Instead of specifying the frequency in hours, an enum could be provided to specify hourly, daily, weekly or monthly frequency. This was discarded because it wouldn't allow expressions like: "Repeat every two days", and "Repeat every three months".
3. fixed interval vs. minimum interval: Since the feature doesn't provide any guarantees of when the periodic task will be run, the API takes a minimum_interval during registration rather than a specified repetition period.
4. Support specifying expire_after when enabling a periodic sync. This isn't being considered, at least for v1, because there is a way to disable periodic sync for the origin.
5. Provide the web app a way to query the last time a periodic task successfully ran for a registration, via a method on SyncManager. This was discarded because using the app-specific logic to keep track of the last time state was updated is better. For instance, consider:
   7am: Periodic sync happens, and the app downloads "Edition 4".

   9am: I visit the app, and get "Edition 4" from the cache. But, "Edition 5" is also out now, so the app downloads and caches it.

11am: Periodic sync happens.

In the second periodic sync, the "last periodic sync" time will be 7am, which the app could use to conclude "We should download and cache Edition 5", but that's already downloaded.

In this case, app specific logic is much better. "What's the latest edition the user saw, vs what's the latest edition available".

References

1. Alternatives considered, with a list of pros and cons.
2. Jake's document on one-periodic sync per origin vs multiple.
3. Periodic Background Sync Implementation Design.