**Chrome. Alarm (API)**

**Question:**

Explain this chrome. Alarms API in detail:

Home

Docs

Chrome Extensions

Reference

API

Was this helpful?

chrome.alarms

bookmark\_border

Chrome 120: Starting in Chrome 120, the minimum alarm interval has been reduced from 1 minute to 30 seconds. For an alarm to trigger in 30 seconds, set periodInMinutes: 0.5.

Chrome 117: Starting in Chrome 117, the number of active alarms is limited to 500. Once this limit is reached, chrome.alarms.create() will fail. When using a callback, chrome.runtime.lastError will be set. When using promises, the promise will be rejected.

Use the chrome.alarms API to schedule code to run periodically or at a specified time in the future.

Permissions

alarms

To use the chrome.alarms API, declare the "alarms" permission in the manifest

{

"name": "My extension",

...

"permissions": [

"alarms"

],

...

}

Concepts and usage

To ensure reliable behavior, it is helpful to understand how the API behaves.

Device sleep

Alarms continue to run while a device is sleeping. However, an alarm will not wake up a device. When the device wakes up, any missed alarms will fire. Repeating alarms will fire at most once and then be rescheduled using the specified period starting from when the device wakes, not taking into account any time that has already elapsed since the alarm was originally set to run.

Persistence

Alarms generally persist until an extension is updated. However, this is not guaranteed, and alarms may be cleared when the browser is restarted. Consequently, consider setting a value in storage when an alarm is created, and then ensure it exists each time your service worker starts up. For example:

const STORAGE\_KEY = "user-preference-alarm-enabled";

async function checkAlarmState() {

const { alarmEnabled } = await chrome.storage.get(STORAGE\_KEY);

if (alarmEnabled) {

const alarm = await chrome.alarms.get("my-alarm");

if (!alarm) {

await chrome.alarms.create({ periodInMinutes: 1 });

}

}

}

checkAlarmState();

Examples

The following examples show how to use and respond to an alarm. To try this API, install the Alarm API example from the chrome-extension-samples repository.

Set an alarm

The following example sets an alarm in the service worker when the extension is installed:

service-worker.js:

chrome.runtime.onInstalled.addListener(async ({ reason }) => {

if (reason !== 'install') {

return;

}

// Create an alarm so we have something to look at in the demo

await chrome.alarms.create('demo-default-alarm', {

delayInMinutes: 1,

periodInMinutes: 1

});

});

Respond to an alarm

The following example sets the action toolbar icon based on the name of the alarm that went off.

service-worker.js:

chrome.alarms.onAlarm.addListener((alarm) => {

chrome.action.setIcon({

path: getIconPath(alarm.name),

});

});

Types:

Alarm

Properties

name

string

Name of this alarm.

periodInMinutes

number optional

If not null, the alarm is a repeating alarm and will fire again in periodInMinutes minutes.

scheduledTime

number

Time at which this alarm was scheduled to fire, in milliseconds past the epoch (e.g. Date.now() + n). For performance reasons, the alarm may have been delayed an arbitrary amount beyond this.

AlarmCreateInfo

Properties

delayInMinutes

number optional

Length of time in minutes after which the onAlarm event should fire.

periodInMinutes

number optional

If set, the onAlarm event should fire every periodInMinutes minutes after the initial event specified by when or delayInMinutes. If not set, the alarm will only fire once.

when

number optional

Time at which the alarm should fire, in milliseconds past the epoch (e.g. Date.now() + n).

Methods

clear()

Promise

chrome.alarms.clear(

name?: string,

callback?: function,

)

Clears the alarm with the given name.

Parameters

name

string optional

The name of the alarm to clear. Defaults to the empty string.

callback

function optional

The callback parameter looks like:

(wasCleared: boolean) => void

wasCleared

boolean

Returns

Promise<boolean>

Chrome 91+

Promises are supported in Manifest V3 and later, but callbacks are provided for backward compatibility. You cannot use both on the same function call. The promise resolves with the same type that is passed to the callback.

clearAll()

Promise

chrome.alarms.clearAll(

callback?: function,

)

Clears all alarms.

Parameters

callback

function optional

The callback parameter looks like:

(wasCleared: boolean) => void

wasCleared

boolean

Returns

Promise<boolean>

Chrome 91+

Promises are supported in Manifest V3 and later, but callbacks are provided for backward compatibility. You cannot use both on the same function call. The promise resolves with the same type that is passed to the callback.

create()

Promise

chrome.alarms.create(

name?: string,

alarmInfo: AlarmCreateInfo,

callback?: function,

)

Creates an alarm. Near the time(s) specified by alarmInfo, the onAlarm event is fired. If there is another alarm with the same name (or no name if none is specified), it will be cancelled and replaced by this alarm.

In order to reduce the load on the user's machine, Chrome limits alarms to at most once every 30 seconds but may delay them an arbitrary amount more. That is, setting delayInMinutes or periodInMinutes to less than 0.5 will not be honored and will cause a warning. when can be set to less than 30 seconds after "now" without warning but won't actually cause the alarm to fire for at least 30 seconds.

To help you debug your app or extension, when you've loaded it unpacked, there's no limit to how often the alarm can fire.

Parameters

name

string optional

Optional name to identify this alarm. Defaults to the empty string.

alarmInfo

AlarmCreateInfo

Describes when the alarm should fire. The initial time must be specified by either when or delayInMinutes (but not both). If periodInMinutes is set, the alarm will repeat every periodInMinutes minutes after the initial event. If neither when or delayInMinutes is set for a repeating alarm, periodInMinutes is used as the default for delayInMinutes.

callback

function optional

Chrome 111+

The callback parameter looks like:

() => void

Returns

Promise<void>

Chrome 111+

Promises are supported in Manifest V3 and later, but callbacks are provided for backward compatibility. You cannot use both on the same function call. The promise resolves with the same type that is passed to the callback.

get()

Promise

chrome.alarms.get(

name?: string,

callback?: function,

)

Retrieves details about the specified alarm.

Parameters

name

string optional

The name of the alarm to get. Defaults to the empty string.

callback

function optional

The callback parameter looks like:

(alarm?: Alarm) => void

alarm

Alarm optional

Returns

Promise<Alarm | undefined>

Chrome 91+

Promises are supported in Manifest V3 and later, but callbacks are provided for backward compatibility. You cannot use both on the same function call. The promise resolves with the same type that is passed to the callback.

getAll()

Promise

chrome.alarms.getAll(

callback?: function,

)

Gets an array of all the alarms.

Parameters

callback

function optional

The callback parameter looks like:

(alarms: Alarm[]) => void

alarms

Alarm[]

Returns

Promise<Alarm[]>

Chrome 91+

Promises are supported in Manifest V3 and later, but callbacks are provided for backward compatibility. You cannot use both on the same function call. The promise resolves with the same type that is passed to the callback.

Events

onAlarm

chrome.alarms.onAlarm.addListener(

callback: function,

)

Fired when an alarm has elapsed. Useful for event pages.

Parameters

callback

function

The callback parameter looks like:

(alarm: Alarm) => void

alarm

Alarm

**ChatGPT’s Answer**

**Detailed Explanation of chrome.alarms API**

The chrome.alarms API allows Chrome extensions to schedule code execution at a specified time or periodically. This is particularly useful for background tasks that need to run at regular intervals without relying on user interaction.

**1. Permissions**

To use the chrome.alarms API, you must declare the alarms permission in the extension's manifest.json file:

{

"name": "My Extension",

...

"permissions": ["alarms"],

...

}

This permission enables the extension to create and manage alarms.

**2. Behavior and Constraints**

**Device Sleep**

* Alarms continue to exist while the device is sleeping but do **not wake up** the device.
* Missed alarms fire once the device wakes up.
* **Repeating alarms** fire at most **once** after waking up and are then rescheduled based on the wake-up time, not the original time.

**Persistence**

* Alarms generally **persist** until an extension is updated.
* **Not guaranteed** to persist across browser restarts.
* Best practice: Store alarm state in chrome.storage and recreate it on extension startup.

Example:

Javascript

const STORAGE\_KEY = "user-preference-alarm-enabled";

async function checkAlarmState() {

const { alarmEnabled } = await chrome.storage.local.get(STORAGE\_KEY);

if (alarmEnabled) {

const alarm = await chrome.alarms.get("my-alarm");

if (!alarm) {

await chrome.alarms.create("my-alarm", { periodInMinutes: 1 });

}

}

}

checkAlarmState();

This ensures the alarm is recreated if it was lost.

**3. Creating and Managing Alarms**

**Creating an Alarm**

To create an alarm, use chrome.alarms.create().

**Example: Set an alarm**

chrome.alarms.create("demo-alarm", {

delayInMinutes: 1, // Fires after 1 minute

periodInMinutes: 5 // Repeats every 5 minutes

});

 delayInMinutes: Specifies how long (in minutes) after creation the alarm should first fire.

 periodInMinutes: If set, the alarm repeats every n minutes.

**Responding to an Alarm**

Alarms trigger the chrome.alarms.onAlarm event.

**Example: Execute code when an alarm fires**

chrome.alarms.onAlarm.addListener((alarm) => {

console.log(`Alarm fired: ${alarm.name}`);

});

This listens for any alarm and logs its name when it fires.

**Retrieving Alarms**

To check for existing alarms:

**Get a specific alarm**

chrome.alarms.get("demo-alarm", (alarm) => {

console.log(alarm);

});

If the alarm exists, it returns an Alarm object.

**Get all alarms**

javascript

chrome.alarms.getAll((alarms) => {

console.log(alarms);

});

Returns an array of all active alarms.

**Clearing Alarms**

You can remove alarms using clear() and clearAll().

**Clear a specific alarm**

chrome.alarms.clear("demo-alarm", (wasCleared) => {

console.log(`Alarm cleared: ${wasCleared}`);

});

**4. Alarm Properties**

When an alarm is retrieved, it contains the following properties:

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Description** |
| name | string | Name of the alarm |
| periodInMinutes | number (optional) | Interval for repeating alarms |
| scheduledTime | number | Scheduled time in milliseconds past epoch (e.g., Date.now() + n) |

**5. Advanced Alarm Configuration**

Alarms can be scheduled based on a **specific timestamp**.

**Example: Set an alarm for a specific time**

const futureTime = Date.now() + 60000; // 1 minute from now

chrome.alarms.create("future-alarm", {

when: futureTime

});

Here, when specifies an absolute time (milliseconds since epoch).

**6. API Methods Overview**

|  |  |
| --- | --- |
| **Method** | **Description** |
| create(name, alarmInfo) | Creates a new alarm (replaces existing alarm with the same name). |
| clear(name) | Clears an alarm by name. |
| clearAll() | Clears all alarms. |
| get(name) | Retrieves details about a specific alarm. |
| getAll() | Gets all active alarms. |

**7. Event Handling**

|  |  |
| --- | --- |
| **Event** | **Description** |
| onAlarm | Fired when an alarm goes off. |

**Example: Update an icon when an alarm fires**

chrome.alarms.onAlarm.addListener((alarm) => {

chrome.action.setIcon({ path: getIconPath(alarm.name) });

});

This changes the extension icon based on which alarm fired.

**8. Limits & Constraints**

* **Minimum Interval:** Starting **Chrome 120**, the minimum period is **30 seconds** (periodInMinutes: 0.5).
* **Maximum Alarms:** Starting **Chrome 117**, extensions can only have **500 active alarms**.
* **Background Execution:** Alarms will fire when the device wakes but will **not wake the device**.

**9. Summary**

* **Use chrome.alarms.create()** to set alarms with a delay or a periodic trigger.
* **Use chrome.alarms.onAlarm.addListener()** to handle alarm triggers.
* **Alarms persist until an extension is updated or the browser restarts** (not guaranteed).
* **Use chrome.alarms.get() and chrome.alarms.getAll()** to check alarm status.
* **Use chrome.alarms.clear() and chrome.alarms.clearAll()** to remove alarms.

This API is useful for scheduling tasks such as:

* Fetching new data periodically
* Running background maintenance tasks
* Reminding users of an event
* Updating UI elements at regular intervals

Would you like a specific example or use case breakdown? 🚀

**You said:**

yes

**ChatGPT said:**

**Use Case Examples for chrome.alarms API 🚀**

Here are **three practical use cases** demonstrating how to use the chrome.alarms API effectively in a Chrome extension.

**1️⃣ Use Case: Periodic Data Fetching (News Updates)**

**Scenario:**

A Chrome extension fetches the latest news from an API every 5 minutes and displays a notification.

**Implementation:**

**1. Set Up the Alarm in background.js**

chrome.runtime.onInstalled.addListener(() => {

chrome.alarms.create("fetch-news", { periodInMinutes: 5 });

});

This creates a repeating alarm that fires every 5 minutes.

**2. Fetch News Data When the Alarm Fires**

chrome.alarms.onAlarm.addListener((alarm) => {

if (alarm.name === "fetch-news") {

fetchLatestNews();

}

});

async function fetchLatestNews() {

try {

const response = await fetch("https://newsapi.org/v2/top-headlines?country=us&apiKey=YOUR\_API\_KEY");

const data = await response.json();

// Show a notification with the latest headline

chrome.notifications.create({

type: "basic",

iconUrl: "icon.png",

title: "Latest News",

message: data.articles[0].title

});

} catch (error) {

console.error("Error fetching news:", error);

}

}

**How it Works**

✅ The extension sets an alarm on installation.  
✅ Every 5 minutes, the alarm fires and fetches news from the API.  
✅ A Chrome notification displays the latest headline.

**2️⃣ Use Case: Reminder Notifications**

**Scenario:**

A user sets a reminder in the extension, and a notification pops up at the scheduled time.

**Implementation:**

**1. Create the Alarm When User Sets a Reminder**

function setReminder(timeInMinutes) {

chrome.alarms.create("user-reminder", { delayInMinutes: timeInMinutes });

}

When the user inputs a reminder time (e.g., 10 minutes), the alarm is set to fire in **10 minutes**.

**2. Show a Notification When Reminder Fires**

chrome.alarms.onAlarm.addListener((alarm) => {

if (alarm.name === "user-reminder") {

chrome.notifications.create({

type: "basic",

iconUrl: "icon.png",

title: "Reminder",

message: "Hey! This is your reminder!"

});

}

});

**How it Works**

✅ The user sets a reminder time.  
✅ The alarm fires at the scheduled time.  
✅ A Chrome notification reminds the user.

**3️⃣ Use Case: Automatic Tab Closing**

**Scenario:**

A user opens a temporary tab, and the extension automatically closes it after 2 minutes.

**Implementation:**

**1. Set an Alarm When a New Tab Opens**

chrome.tabs.onCreated.addListener((tab) => {

chrome.alarms.create(`close-tab-${tab.id}`, { delayInMinutes: 2 });

});

Every time a new tab is opened, an alarm is created for that tab.

**2. Close the Tab When the Alarm Fires**

chrome.alarms.onAlarm.addListener((alarm) => {

if (alarm.name.startsWith("close-tab-")) {

const tabId = parseInt(alarm.name.split("-")[2]); // Extract tab ID

chrome.tabs.remove(tabId);

}

});

**How it Works**

✅ Each tab gets a **2-minute** countdown before closing.  
✅ When the alarm fires, the extension **closes the tab** automatically.

**Bonus: Stopping All Alarms**

If you want a "Stop All" button to cancel all alarms:

function stopAllAlarms() {

chrome.alarms.clearAll(() => {

console.log("All alarms cleared!");

});

}

**Final Thoughts**

The chrome.alarms API is powerful for automating background tasks. You can:

* ✅ Fetch data periodically (e.g., news updates, weather reports)
* ✅ Notify users of reminders
* ✅ Automate tab management