Collab Chrome Extension

The anatomy of a Daily video call Chrome extension

Understanding Chrome extension APIs and how to integrate Daily's core APIs



Jess Mitchell
Published: 13 Aug 2021 (Updated: 14 Jan 2022) • 8 min read



> *[14/01/22] This demo app uses manifest v2. Manifest v2 is being deprecated by Chrome in 2022 and will no longer be accepted into the Chrome store.*

At Daily, we talk a lot about Chrome extensions because they're such a flexible way to quickly add a video call to your web-browsing experience. Previously, we looked at how to update Daily Prebuilt's UI

using a Chrome extension, as well as how to launch a Daily screen share from any website.

More recently, we built a Notion integration for Daily video calls called Daily Collab, which allows you to start a Daily call in any Notion doc and transcribe it as you chat. (Pretty cool, right?)

The possibilities are endless for how Daily can be used in Chrome extensions. To help our customers feel more confident about building out new Daily Collab features or even starting from scratch, this tutorial will cover the anatomy of Chrome extensions in general. We'll be using the Daily Collab demo as the main example for understanding how all the different pieces of an extension fit together.

A word on manifest versions

Chrome released Manifest v3 for Chrome extensions in November 2020 to help improve previous security risks. One of the biggest changes in v3 is disallowing a Chrome extension from downloading remote code.

This presents a bit of an issue for how `daily-js` currently works but, thankfully, it will be resolved in future versions.

For most of the following material, the manifest version being used doesn't impact how the extension is written. The majority of the content will be the same whether you're using v2 or v3.

Where there are differences—namely, the `manifest.json` file and background script— we'll explicitly name which information applies to v2 or v3.

In terms of what that means for the Daily Collab demo code specifically, the most important detail to note is that we used a v2 background script instead of converting it to a v3 background service worker.

To learn more about the changes between v2 and v3, check out Google's migration guide.

What's on the agenda today

To better understand what you can build with a Chrome extension, we'll review the main components of Chrome extensions, as well as some additional general information.

Note that you don't need to use all these elements to build a Chrome extension, but it's good to know what is available.

The possibilities for what you can do with a Chrome extension are quite extensive so in this overview we'll cover the pieces the Daily Collab Chrome extension demo uses:

- The `manifest.json` file, which is mandatory for all Chrome extensions. (Note: Migrating to v3 will require changes to this file)
- The Chrome extension popup menu
- Background scripts. (Note: In v3, this will be a service worker instead)
- Content scripts
- Browser messages
- Accessing individual tabs

Understanding the extension's manifest

Each Chrome extension has a manifest represented by the `manifest.json` file. This JSON file should be located at the root of your extension's directory.

```
{
 "name": "My New Chrome Extension",
 "version": "1.0",
 "description": "This extension does something!",
 "manifest_version": 3
}
```

The manifest instructs your browser as to what the extension can (and cannot) do, such as which parts of the browser the extension has access

to, and which scripts it's permitted to run. For example, if the Chrome extension needs to access information on the browser's tab, `tabs` must be added to the manifest's `permissions` field. Similarly, to run content scripts on web pages, all the URLs where the content scripts are permitted to run must be added to the `manifest.json` file.

The Daily Collab manifest, for example, includes permissions for accessing tabs, as well as most Notion URLs.

```
"manifest_version": 2,
 "version": "0.0.0.1",
 "name": "Daily Collab",
 "description": "Embed and transcribe video calls directly in yo
 "background": { "service_worker": "background.bundle.js" },
 "icons": {
   "128": "icon-128.png"
 },
 "browser_action": {
   "default_popup": "popup.html",
   "default_icon": "icon-34.png"
 },
 "permissions": ["https://daily-notion-api.vercel.app/*", "tabs"
 "content_scripts": [
   {
     "matches": ["https://www.notion.so/*"],
     "exclude_matches": [
       "https://www.notion.so/onboarding",
       "https://www.notion.so/install-integration/*",
       "https://www.notion.so/my-integrations/*"
     ],
     "js": ["contentScript.bundle.js"]
   }
 ],
 ...
}
```

Background service workers/scripts: Adding functionality to your Chrome extension

As mentioned, background scripts are used in v2 of Chrome extensions and background service workers are used in v3. Overall, the actual code

being written in both will look fairly similar.
To start, let's see what these both have in common.

Regardless of your manifest version

Background service workers and scripts both communicate with other parts of the extension via message passing. They're like the brain of the extension that can work independently of any user interactions and outside of the context of a specific web page.

They can also access browser tab information, as well as being able to set "alarms" to trigger an action on an interval. They're an incredibly powerful way to run code related to your Chrome extensions functionality while staying "behind the scenes".

One thing to keep in mind, though, is they cannot access the DOM. This is because, as mentioned, they are not run in the context of a specific web page. To access the DOM, you'll need to introduce a content script.

Manifest V2: Background scripts

Background scripts are JavaScript files that run once a Chrome extension is installed in the browser. They are a feature of Manifest V2 and have been replaced by service workers in Manifest V3. Once they become active in the browser, they will run in the background for as long as the browser is open.

Since they are always running, background scripts can have global variables that are updated as different events occur without losing context.

Manifest V3: Background service workers

Background service workers replaced background scripts in Manifest V3. A key difference with a service worker is that they are terminated when not in use then restarted when an event is triggered. This means you cannot include global variables in your service worker; they will

essentially “reset” to the initial declaration each time the service worker is re-initialized.

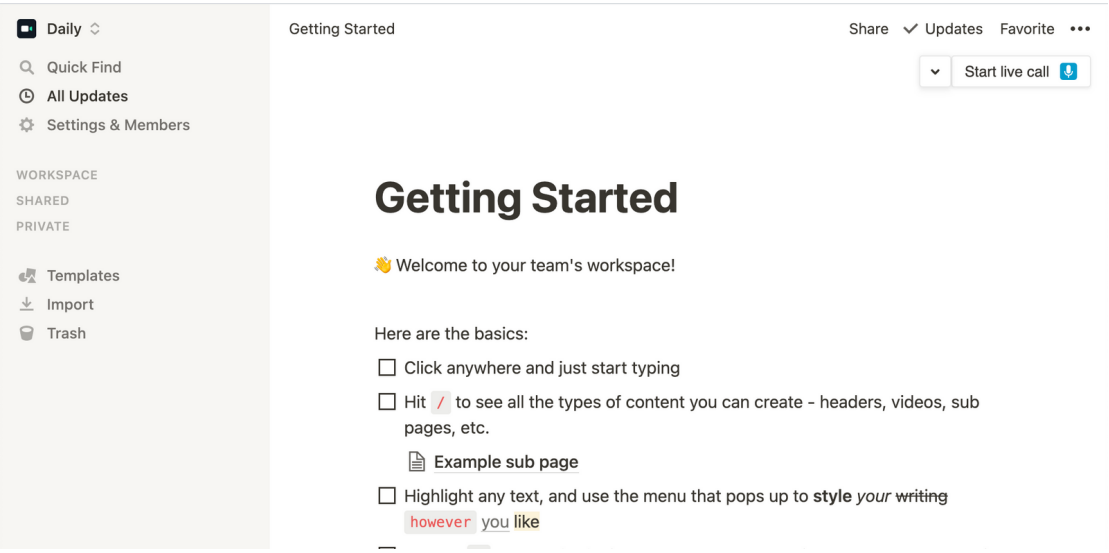
Content scripts: Reading or injecting HTML into the DOM

For Chrome extensions that need to access the DOM of specific webpages, content scripts are required. They run in the context of permitted webpages, i.e. ones that have been permitted in the extension’s manifest.

Unlike background scripts, content scripts can access and read a webpage’s DOM, as well as inject new DOM elements. This capability is useful for features that depend on what is actually present on the page.

Updating the DOM with content scripts can range quite a bit in complexity. You can inject just an element by selecting an existing element in the DOM and appending a new element you’ve created. Alternatively you can inject an entire HTML document from a content script. They’re so flexible, in fact, you can even inject an entire React or Vue app into the page you’re viewing with content scripts.

In Daily Collab, for example, the extension UI you see in Notion is all injected through a content script. It’s actually an entire React app that gets prepended to the DOM’s body element.



Tip: To see a template for how to inject a React app into the page, check out this boilerplate repo on Github to see how it's done.

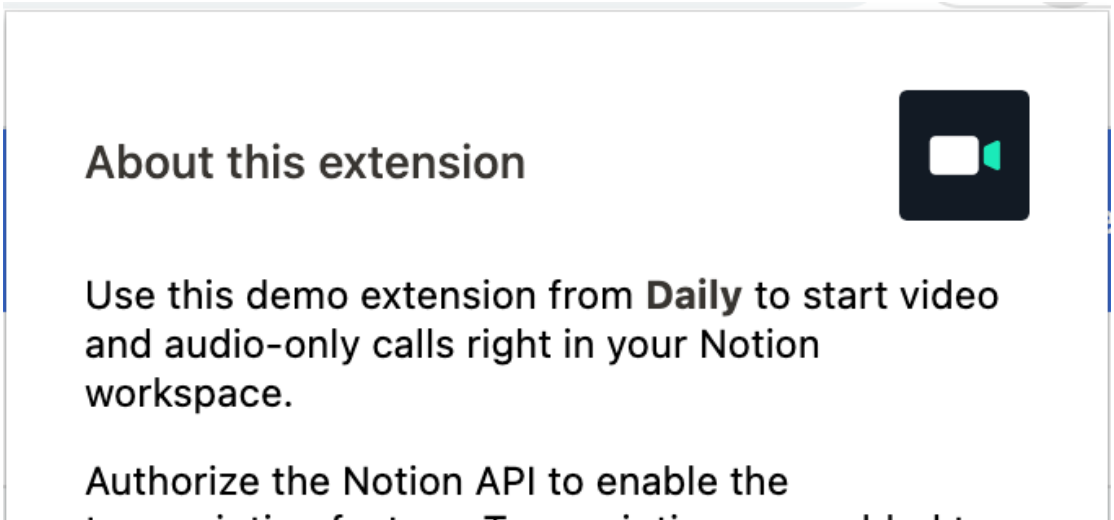
Content scripts can also run JavaScript in the webpage without making any DOM changes. For example, you may need to read from the DOM and update the extension's popup menu based on what's present. To do this, you could run a JavaScript function that reads the DOM and sends a message to the popup based on the DOM's contents.

Overall, the main distinguishing factor of when to use content scripts is when you need to access the DOM of a specific webpage.

Adding a popup menu: Webpage-agnostic user interactions

Chrome extensions often bring to mind the popup button that gets added to the browser's navigation bar once they've been installed. This popup provides a space for users to learn information about the Chrome extension or, more importantly, to have user interactions that don't interfere with webpage content.

In the case of Daily Collab, we used the popup menu as a place just for general extension information. No action required!

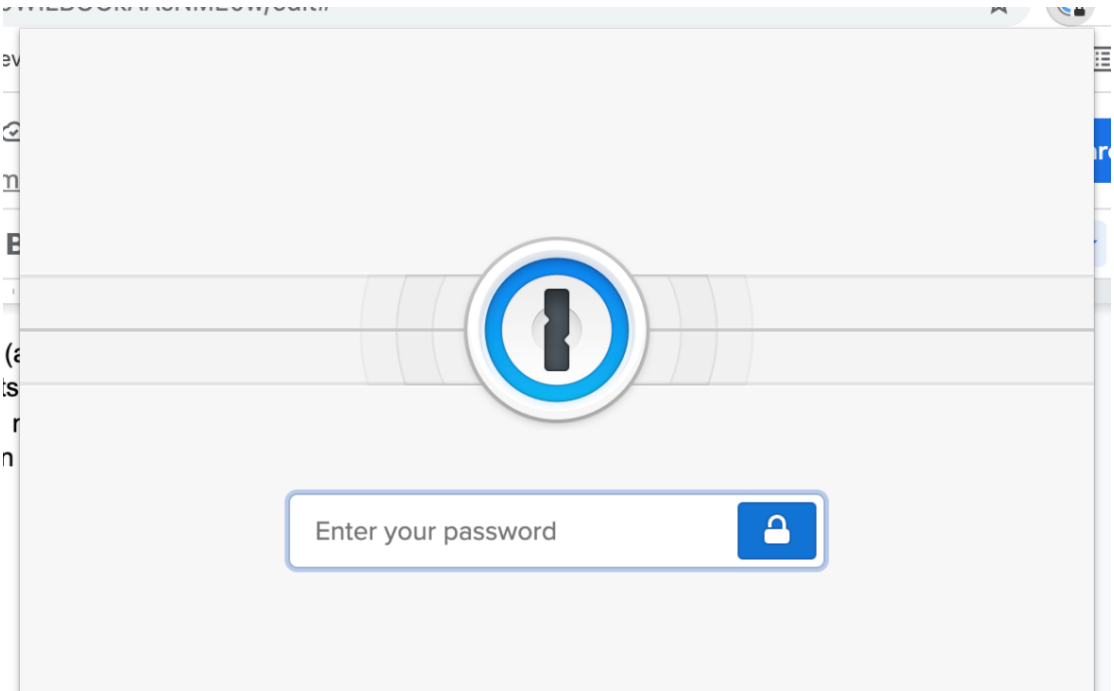


transcription feature. Transcriptions are added to the bottom of the Notion page being currently viewed when transcription is in progress.

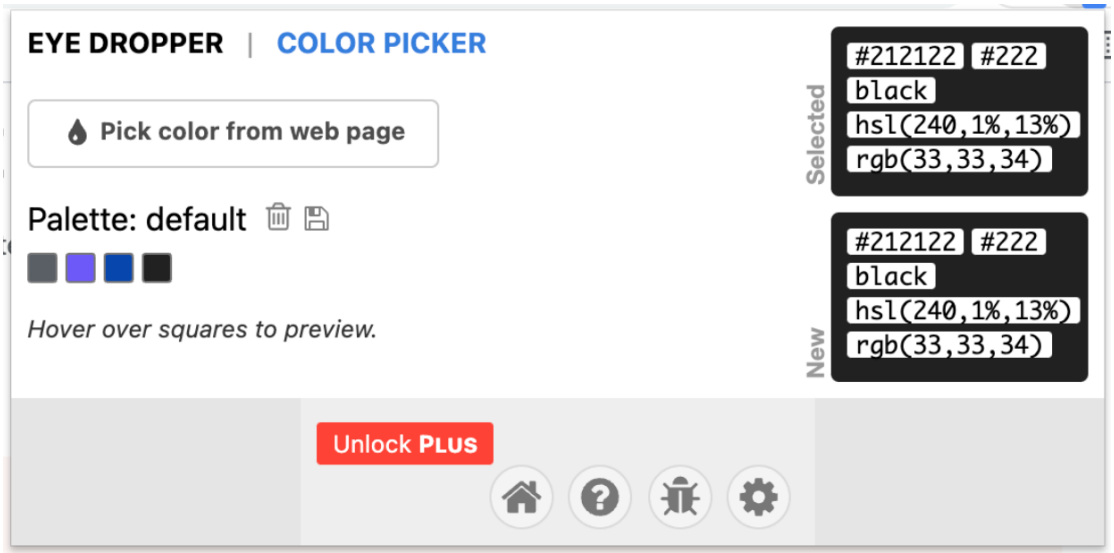
If you installed the extension after opening the Notion doc, be sure to refresh your page.

Daily Collab popup menu

1Password's Chrome extension is another example that shows the usefulness of popup menus. It allows users to log in and access their 1Password account without actually having to visit the 1Password website. This functionality is incredibly helpful for tools that need authorization or some type of user interaction unrelated to the current web page.



An eyedropper extension is another great example of when to use a Chrome extension popup. The user can get information about the colors used in the active webpage and have those colors displayed in the popup even when they have left the page. This is all achieved without ever altering the web page itself.



If you've ever had to get the exact hexadecimal values for a website's color palette, you'll know how useful eyedropper extensions can be!

Tabs: Accessing open webpages

Being able to access browser tabs is another extremely useful tool provided by Chrome extensions.

In the Daily Collab demo, the Tabs API is used to make sure the active tab is a Notion page and to be certain we are correctly tracking which Notion docs have live Daily calls in them.

In this use case, knowing which tab is being viewed is central for knowing which information to have displayed by the content script.

```
chrome.tabs.onUpdated.addListener(function () {
  setTimeout(async () => {
```

```
    await chrome.tabs.query(
      { active: true, currentWindow: true },
      async function (tabs) {
        const tab = tabs[0];
        getDailyRoomForContentScript(tab);
      }
    );
  }, 250);
});
```

Messaging: Communicating between webpages and the extension

Chrome's Messaging API is one of the key features of Chrome extensions that lets most of the aforementioned elements communicate with each other.

A background service worker/script can send a message to a content

script and popup, and vice versa. This provides an extremely effective way to asynchronously update the different parts of your Chrome extension.

To understand how to use the Messaging API, let's look at an example in the Daily Collab code.

In the background script, we listen for any messages sent from other parts of the extension and pass the `handleMessageReceived` callback to determine what action to take based on the message that was sent. For example, if the content script sends a message that includes `createCall` , a new Daily room is created by the background script in response.

```
chrome.runtime.onMessage.addListener(handleMessageReceived);
```

Once the new Daily room is created and available for use, the background script sends a message back to the content script with the Daily room information:

```
async function handleMessageReceived(request, sender, sendRespon
  const {
```

```
  const {
    createCall,
  } = request;

  ...
  if (createCall) {
    await createDailyCall(request, sender.tab);
  }

  ...
  sendResponse('message received');
}

function createDailyCall(request) {
  const notionId = getNotionId(tab);
  if (!notionId) return;
  const room = await createRoom({
    baseUrl: BASE_URL,
    tabId: notionId,
    audioOnly: request.audioOnly,
    useTranscription: request.useTranscription,
    workspaceId: request?.workspaceId || null,

    docUrl: tab?.url,
  });

   const message = room.dailyUrl
       ? {
           dailyUrl,
           audioOnly: room.audioOnly,
           isTranscribing: room.isTranscribing,
           tabId: tab.id,
           tabChange: false,
           useTranscription: room.useTranscription,
         }
       : { error: 'Room could not be created' };


   // Send a message back to the content script with the room in
   chrome.tabs.sendMessage(tab.id, message, noop);
   ...
}
```

By sending information through messages, content scripts, background scripts/service workers, and pop ups can all communicate and request information they may otherwise not have access to.

Wrapping up

We hope this clarifies some of the various areas of how to build out Chrome extension features. Having a thorough understanding of all your Chrome extensions options will help as we move through our Daily Collab Chrome extension series, where we'll look at building out the extension functionality to embed Daily calls in Notion pages and transcribe the call's audio.

To get a head start, be sure to check out our previous post in the Daily Collab blog post series, as well as the Daily Collab demo's source code.

And, as always, let us know if you have any questions!

Happy coding!

Get updates from Daily

Don't miss a thing! Subscribe to our monthly newsletter and we'll keep you up-to-date with the latest product releases, tutorials, and more from Daily.

Email*

Country*

Join peerConnection - our video/WebRTC developer community.

By clicking submit below, you consent to allow Daily.co to store and process the personal information submitted above to provide you the content requested.

Submit

More in **Collab Chrome Extension**

Build your own "prejoin" call UI in a custom Daily video chat app

9 Sep 2021 - 9 min read

Tutorial: Append Daily call transcription text with the Notion REST API

2 Sep 2021 - 8 min read

Tutorial: Build a custom Daily video call widget and embed it into any Notion page

26 Aug 2021 - 11 min read

See all 5 posts →



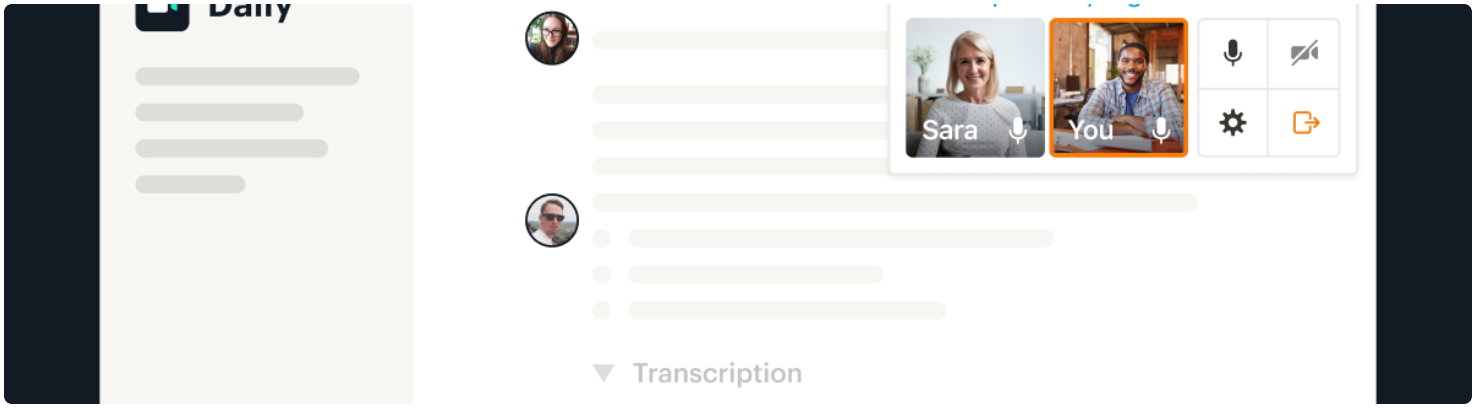
Collab Chrome Extension

Authorizing Daily's Chrome extension transcription feature with Notion's API

Using the Notion API's internal and public integration to authenticate Notion document edits



Jess Mitchell
19 Aug 2021 – 8 min read



Collab Chrome Extension

Embed and transcribe a Daily WebRTC video call in any Notion page

Introducing Daily Collab, a Chrome extension demo using Daily's WebRTC video chat APIs, Notion's beta API, and Deepgram transcription



Jess Mitchell
13 Aug 2021 – 5 min read