Inject scripts into the active tab

Simplify the styling of the current page by clicking the extension toolbar icon.

Overview

This tutorial builds an extension that simplifies the styling of the Chrome extension and Chrome Web Store documentation pages so that they are easier to read.

In this guide, we're going to explain how to do the following:

- Use the extension service worker as the event coordinator.
- Preserve user privacy through the `"activeTab"` permission.
- Run code when the user clicks the extension toolbar icon.
- Insert and remove a style sheet using the Scripting API.
- Use a keyboard shortcut to execute code.

Before you start

This guide assumes that you have basic web development experience. We recommend checking out Hello World for an introduction to the extension development workflow.

Build the extension

To start, create a new directory called `focus-mode` that will hold the extension's files. If you prefer, you can download the complete source code from GitHub.

Step 1: Add the extension data and icons

Create a file called `manifest.json` and include the following code.

```
{
  "manifest_version": 3,
  "name": "Focus Mode",
  "description": "Enable focus mode on Chrome's official Extensions and Chrome
Web Store documentation.",
  "version": "1.0",
  "icons": {
    "16": "images/icon-16.png",
    "32": "images/icon-32.png",
    "48": "images/icon-48.png",
    "128": "images/icon-128.png"
  }
}
```

To learn more about these manifest keys, check out the "Run scripts on every tab" tutorial that explains the extension's metadata and icons in more detail.

Create an `images` folder then download the icons into it.

Step 2: Initialize the extension

Extensions can monitor browser events in the background using the extension's service worker. Service workers are special JavaScript environments that handle events and terminate when they're not needed.

Start by registering the service worker in the `manifest.json` file:

```
{
  ...
  "background": {
    "service_worker": "background.js"
  },
  ...
}
```

Create a file called `background.js` and add the following code:

```
chrome.runtime.onInstalled.addListener(() => {
  chrome.action.setBadgeText({
    text: "OFF",
  });
});
```

The first event our service worker will listen for is `runtime.onInstalled()`. This method allows the extension to set an initial state or complete some tasks on installation. Extensions can use the Storage API and IndexedDB to store the application state. In this case, though, since we're only handling two states, we will use the *action's badge* text itself to track whether the extension is 'ON' or 'OFF'.

Key term: The action's badge is a colored banner on top of the extension action (toolbar icon).

Step 3: Enable the extension action

The *extension action* controls the extension's toolbar icon. So whenever the user clicks the extension icon, it will either run some code (like in this example) or display a popup. Add the following code to declare the extension action in the `manifest.json` file:

```
{
  ...
  "action": {
    "default_icon": {
      "16": "images/icon-16.png",
      "32": "images/icon-32.png",
      "48": "images/icon-48.png",
      "128": "images/icon-128.png"
    }
  },
  ...
}
```

Use the activeTab permission to protect user privacy

The `activeTab` permission grants the extension *temporary* ability to execute code on the active tab. It also allows access to sensitive properties of the current tab.

This permission is enabled when the user *__invokes__* the extension. In this case, the user invokes the extension by clicking on the extension action.

💡 **What other user interactions enable the activeTab permission in my own extension?**

- Pressing a keyboard shortcut combination.
- Selecting a context menu item.
- Accepting a suggestion from the omnibox.
- Opening an extension popup.

The `"activeTab"` permission allows users to *purposefully* choose to run the extension on the focused tab; this way, it protects the user's privacy. Another benefit is that it does not trigger a permission warning.

To use the `"activeTab"` permission, add it to the manifest's permission array:

```
{
  ...
  "permissions": ["activeTab"],
  ...
}
```

Step 4: Track the state of the current tab

After the user clicks the extension action, the extension will check if the URL matches a documentation page. Next, it will check the state of the current tab and set the next state. Add the following code to `background.js`:

```
const extensions = 'https://developer.chrome.com/docs/extensions'
const webstore = 'https://developer.chrome.com/docs/webstore'

chrome.action.onClicked.addListener(async (tab) => {
  if (tab.url.startsWith(extensions) || tab.url.startsWith(webstore)) {
    // Retrieve the action badge to check if the extension is 'ON' or 'OFF'
    const prevState = await chrome.action.getBadgeText({ tabId: tab.id });
    // Next state will always be the opposite
    const nextState = prevState === 'ON' ? 'OFF' : 'ON'

    // Set the action badge to the next state
    await chrome.action.setBadgeText({
      tabId: tab.id,
      text: nextState,
    });
  }
});
```

Step 5: Add or remove the style sheet

Now it's time to change the layout of the page. Create a file named `focus-mode.css` and include the following code:

```css
body > .scaffold > :is(top-nav, navigation-rail, side-nav, footer),
main > :not(:last-child),
main > :last-child > navigation-tree,
main .toc-container {
  display: none;
}

main > :last-child {
  margin-top: min(10vmax, 10rem);
  margin-bottom: min(10vmax, 10rem);
}
```

Insert or remove the style sheet using the Scripting API. Start by declaring the `"scripting"` permission in the manifest:

```json
{
  ...
  "permissions": ["activeTab", "scripting"],
  ...
}
```

Success: The Scripting API does not trigger a permission warning.

Finally, in `background.js` add the following code to change the layout of the page:

```js
  ...
    if (nextState === "ON") {
      // Insert the CSS file when the user turns the extension on
      await chrome.scripting.insertCSS({
        files: ["focus-mode.css"],
        target: { tabId: tab.id },
      });
    } else if (nextState === "OFF") {
      // Remove the CSS file when the user turns the extension off
      await chrome.scripting.removeCSS({
        files: ["focus-mode.css"],
        target: { tabId: tab.id },
      });
    }
  }
});
```

💡 **Can I use the Scripting API to inject code instead of a style sheet?**

Yes. You can use `scripting.executeScript()` to inject JavaScript.

Optional: Assign a keyboard shortcut

Just for fun, add a shortcut to make it easier to enable or disable focus mode. Add the `"commands"` key to the manifest.

```
{
  ...
  "commands": {
    "_execute_action": {
      "suggested_key": {
        "default": "Ctrl+B",
        "mac": "Command+B"
      }
    }
  }
}
```

The `"_execute_action"` key runs the same code as the `action.onClicked()` event, so no additional code is needed.

Test that it works

Verify that the file structure of your project looks like the following:

```
focus-mode/
├── manifest.json
├── background.js
├── focus-mode.css
└── images/
    ├── icon-16.png
    ├── icon-32.png
    ├── icon-48.png
    └── icon-128.png
```

Load your extension locally

To load an unpacked extension in developer mode, follow the steps in Hello World.

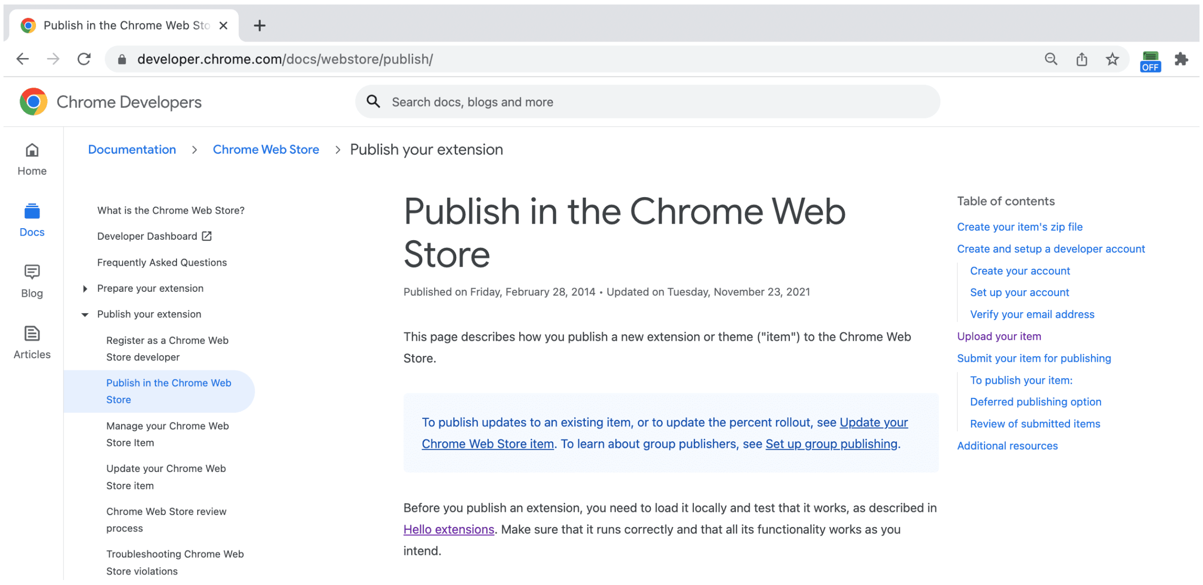
Test the extension on a documentation page

First, open any of the following pages:

- Welcome to the Chrome Extension documentation
- Publish in the Chrome Web Store
- Scripting API

Then, click the extension action. If you set up a keyboard shortcut, you can test it by pressing `Ctrl + B` or `Cmd + B`.

It should go from this:


Publish in the Chrome Web Sto ×
developer.chrome.com/docs/webstore/publish/
OFF
Chrome Developers
Search docs, blogs and more
Home
Docs
Blog
Articles

Documentation > Chrome Web Store > Publish your extension

What is the Chrome Web Store?
Developer Dashboard
Frequently Asked Questions
Prepare your extension
Publish your extension
Register as a Chrome Web Store developer
Publish in the Chrome Web Store
Manage your Chrome Web Store Item
Update your Chrome Web Store item
Chrome Web Store review process
Troubleshooting Chrome Web Store violations

Publish in the Chrome Web Store

Published on Friday, February 28, 2014 • Updated on Tuesday, November 23, 2021

This page describes how you publish a new extension or theme ("item") to the Chrome Web Store.

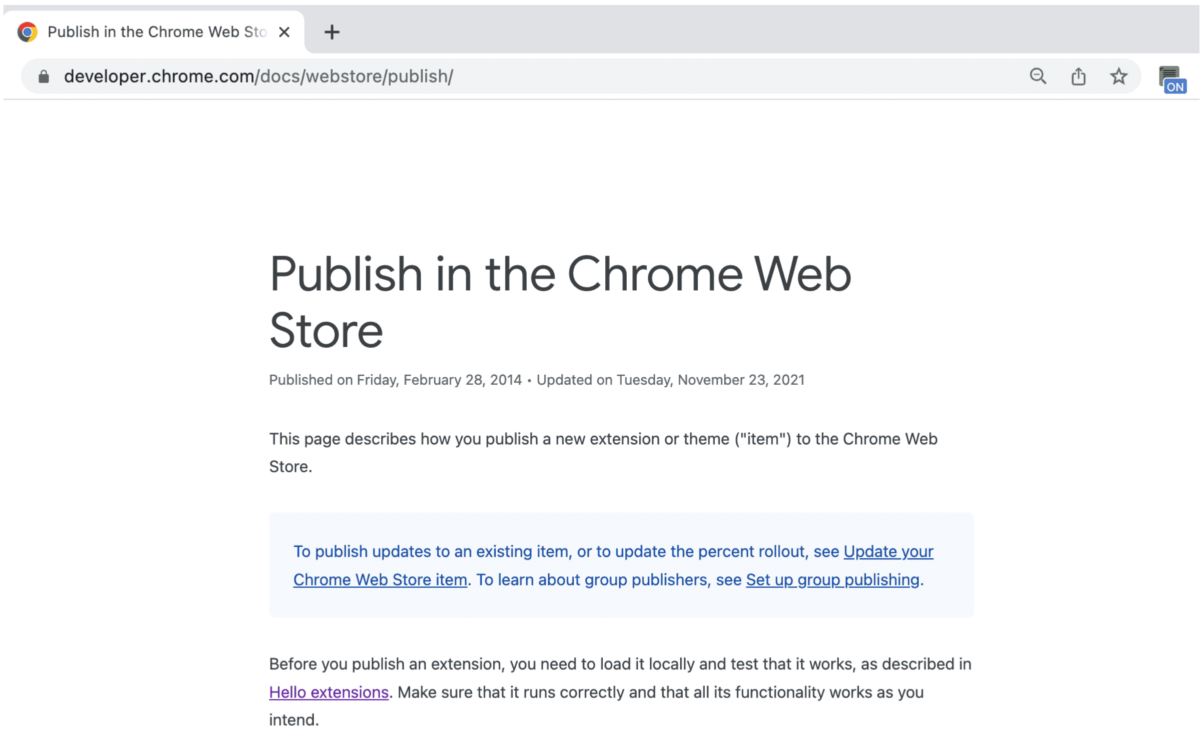
To publish updates to an existing item, or to update the percent rollout, see Update your Chrome Web Store item. To learn about group publishers, see Set up group publishing.

Before you publish an extension, you need to load it locally and test that it works, as described in Hello extensions. Make sure that it runs correctly and that all its functionality works as you intend.

Table of contents
Create your item's zip file
Create and setup a developer account
Create your account
Set up your account
Verify your email address
Upload your item
Submit your item for publishing
To publish your item:
Deferred publishing option
Review of submitted items
Additional resources

Focus Mode extension off

To this:


Publish in the Chrome Web Sto ×
developer.chrome.com/docs/webstore/publish/
ON

Publish in the Chrome Web Store

Published on Friday, February 28, 2014 • Updated on Tuesday, November 23, 2021

This page describes how you publish a new extension or theme ("item") to the Chrome Web Store.

To publish updates to an existing item, or to update the percent rollout, see Update your Chrome Web Store item. To learn about group publishers, see Set up group publishing.

Before you publish an extension, you need to load it locally and test that it works, as described in Hello extensions. Make sure that it runs correctly and that all its functionality works as you intend.

Focus Mode extension on

🎯 Potential enhancements

Based on what you've learned today, try to accomplish any of the following:

- Improve the CSS style sheet.
- Assign a different keyboard shortcut.
- Change the layout of your favorite blog or documentation site.

Keep building.

Congratulations on finishing this tutorial 🎉. Continue leveling up your skills by completing other tutorials on this series:

Extension	What you will learn
Reading time	To insert an element on a specific set of pages automatically.
Tabs Manager	To create a popup that manages browser tabs.

Continue exploring

We hope you enjoyed building this Chrome extension and are excited to continue your extension development learning journey. We recommend the following learning paths:

- The developer's guide has dozens of additional links to pieces of documentation relevant to advanced extension creation.
- Extensions have access to powerful APIs beyond what's available on the open web. The Chrome APIs documentation walks through each API.