Notification behavior

- Notification Click Event
- Actions
  - Buttons
  - Inline Replies
- Tag
- Renotify
- Silent
- Requires interaction
- Where to go next
  - Code labs

So far, we've looked at the options that alter the visual appearance of a notification. There are also options that alter the behavior of notifications.

By default, calling `showNotification()` with just visual options will have the following behaviors:

- Clicking on the notification does nothing.
- Each new notification is shown one after the other. The browser will not collapse the notifications in any way.
- The platform may play a sound or vibrate the user's device (depending on the platform).
- On some platforms, the notification will disappear after a short period of time while others will show the notification unless the user interacts with it. (For example, compare your notifications on Android and Desktop.)

In this section, we are going to look at how we can alter these default behaviors using options alone. These are relatively easy to implement and take advantage of.

Notification Click Event

When a user clicks on a notification, the default behavior is for nothing to happen. It doesn't even close or remove the notification.

The common practice for a notification click is for it to close and perform some other logic (i.e. open a window or make some API call to the application).

To achieve this, you need to add a `'notificationclick'` event listener to our service worker. This will be called whenever a notification is clicked.

```
self.addEventListener('notificationclick', (event) => {
  const clickedNotification = event.notification;
  clickedNotification.close();

  // Do something as the result of the notification click
  const promiseChain = doSomething();
  event.waitUntil(promiseChain);
});
```

As you can see in this example, the notification that was clicked can be accessed as `event.notification`. From this, you can access the notification's properties and methods. In this case, you call its `close()` method and perform additional work.

You need to make use of `event.waitUntil()` to keep the service worker running while your code is busy.

Actions

Actions allow you to create another level of interaction with your users over just clicking the notification.

Buttons

In the previous section, you saw how to define action buttons when calling `showNotification()`:

```
const title = 'Actions Notification';

const options = {
  actions: [
    {
      action: 'coffee-action',
      title: 'Coffee',
      type: 'button',
      icon: '/images/demos/action-1-128x128.png',
```

```
    },
    {
      action: 'doughnut-action',
      type: 'button',
      title: 'Doughnut',
      icon: '/images/demos/action-2-128x128.png',
    },
    {
      action: 'gramophone-action',
      type: 'button',
      title: 'Gramophone',
      icon: '/images/demos/action-3-128x128.png',
    },
    {
      action: 'atom-action',
      type: 'button',
      title: 'Atom',
      icon: '/images/demos/action-4-128x128.png',
    },
  ],
};

registration.showNotification(title, options);
```

If the user clicks an action button, check the `event.action` value in the `noticationclick` event to tell which action button was clicked.

`event.action` will contain the `action` value set in the options. In the example above the `event.action` values would be one of the following: `'coffee-action'`, `'doughnut-action'`, `'gramophone-action'` or `'atom-action'`.

With this we would detect notification clicks or action clicks like so:

```
self.addEventListener('notificationclick', (event) => {
  if (!event.action) {
    // Was a normal notification click
    console.log('Notification Click.');
    return;
  }

  switch (event.action) {
```

```
    case 'coffee-action':
      console.log("User ❤️'s coffee.");
      break;
    case 'doughnut-action':
      console.log("User ❤️'s doughnuts.");
      break;
    case 'gramophone-action':
      console.log("User ❤️'s music.");
      break;
    case 'atom-action':
      console.log("User ❤️'s science.");
      break;
    default:
      console.log(`Unknown action clicked: '${event.action}'`);
      break;
  }
});
```

Inline Replies

Also, in the previous section, you saw how to add an inline reply to the notification:

```
const title = 'Poll';

const options = {
  body: 'Do you like this photo?',
  image: '/images/demos/cat-image.jpg',
  icon: '/images/demos/icon-512x512.png',
  badge: '/images/demos/badge-128x128.png',
  actions: [
    {
      action: 'yes',
      type: 'button',
      title: '👍Yes',
    },
    {
      action: 'no',
      type: 'text',
      title: '👎No (explain why)',
```

```
      placeholder: 'Type your explanation here',
    },
  ],
};

registration.showNotification(title, options);
```

`event.reply` will contain the value typed by the user in the input field:

```
self.addEventListener('notificationclick', (event) => {
  const reply = event.reply;

  // Do something with the user's reply
  const promiseChain = doSomething(reply);
  event.waitUntil(promiseChain);
});
```

Tag

The `tag` option is essentially a string ID that "groups" notifications together, providing an easy way to determine how multiple notifications are displayed to the user. This is easiest to explain with an example.

Let's display a notification and give it a tag, of `'message-group-1'`. We'd display the notification with this code:

```
const title = 'Notification 1 of 3';

const options = {
  body: "With 'tag' of 'message-group-1'",
  tag: 'message-group-1',
};

registration.showNotification(title, options);
```

Which will show our first notification.

First notification with tag of message group 1.

Let's display a second notification with a new tag of `'message-group-2'`, like so:

```
const title = 'Notification 2 of 3';

const options = {
  body: "With 'tag' of 'message-group-2'",
  tag: 'message-group-2',
};

registration.showNotification(title, options);
```

This will display a second notification to the user.

Two notifications where the second tag of message group 2.

Now let's show a third notification but re-use the first tag of `'message-group-1'`. Doing this will close the first notification and replace it with our new notification.

```
const title = 'Notification 3 of 3';

const options = {
  body: "With 'tag' of 'message-group-1'",
  tag: 'message-group-1',
};

registration.showNotification(title, options);
```

Now we have two notifications even though `showNotification()` was called three times.

Two notifications where the first notification is replaced by a third notification.

The `tag` option is simply a way of grouping messages so that any old notifications that are currently displayed will be closed if they have the same tag as a new notification.

A subtlety to using `tag` is that when it replaces a notification, it will do so *without* a sound or vibration.

This is where the `renotify` option comes in.

Renotify

This largely applies to mobile devices at the time of writing. Setting this option makes new notifications vibrate and play a system sound.

There are scenarios where you might want a replacing notification to notify the user rather than silently update. Chat applications are a good example. In this case, you should set `tag` and `renotify` to `true`.

```
const title = 'Notification 2 of 2';

const options = {
  tag: 'renotify',
  renotify: true,
};

registration.showNotification(title, options);
```

If you set `renotify` to `true` on a notification without a `tag`, you'll get the following error: `TypeError: Failed to execute 'showNotification' on 'ServiceWorkerRegistration': Notifications which set the renotify flag must specify a non-empty tag`

Silent

This option allows you to show a new notification but prevents the default behavior of vibration, sound and turning on the device's display.

This is ideal if your notifications don't require immediate attention from the user.

```
const title = 'Silent Notification';

const options = {
  silent: true,
};

registration.showNotification(title, options);
```

Note: If you define both `silent` and `renotify`, `silent` will take precedence.

Requires interaction

Chrome on desktop will show notifications for a set time period before hiding them. Chrome on Android doesn't have this behavior. Notifications are displayed until the user interacts with them.

To force a notification to stay visible until the user interacts with it, add the `requireInteraction` option. This will show the notification until the user dismisses or clicks your notification.

```
const title = 'Require Interaction Notification';

const options = {
  requireInteraction: true,
};

registration.showNotification(title, options);
```

Use this option with consideration. Showing a notification and forcing the user to stop what they are doing to dismiss your notification can be frustrating.

In the next section, we are going to look at some of the common patterns used on the web for managing notifications and performing actions such as opening pages when a notification is clicked.

Where to go next

- Web Push Notification Overview
- How Push Works
- Subscribing a User
- Permission UX
- Sending Messages with Web Push Libraries
- Web Push Protocol
- Handling Push Events
- Displaying a Notification
- Notification Behavior
- Common Notification Patterns
- Push Notifications FAQ
- Common Issues and Reporting Bugs

Code labs

- Build a push notification client
- Build a push notification server

□□□ #3
□ □□□ □□□ 2026-10-31 22:11:25 UTC
□ cocobook-sync □□□ 2026-11-04 17:37:52 UTC