Migrate from Workbox v2 to v3

This guide is focused on breaking changes introduced in Workbox v3, with examples of what changes you'd need to make when upgrading from a Workbox v2 setup.

If you're currently using the legacy `sw-precache`/`sw-toolbox` combination, and are looking to transition to Workbox for the first time, here's a different migration guide which will help.

v3 Background

Workbox's v3 release represents a significant refactoring of the existing codebase. The overarching goals are:

- *Minimize the size of the Workbox.* The amount of service worker runtime code that's downloaded and executed has been reduced. Instead of opting everyone in to a monolithic bundle, only code for the specific features that you're using will be imported at runtime.
- *Workbox has a CDN.* We provide a fully supported, Google Cloud Storage-based CDN hosting as the canonical option for accessing the Workbox runtime libraries, making it easier to get up and running with Workbox.
- *Better Debugging and Logs.* The debugging and logging experience has been vastly improved. Debug logs are enabled by default whenever Workbox is used from a `localhost` origin and all logging and assertions are stripped from the production builds.

```
▼ workbox Welcome to Workbox!
    📖 Read the guides and documentation
       https://developers.google.com/web/tools/workbox/
    ❓ Use the [workbox] tag on StackOverflow to ask questions
       https://stackoverflow.com/questions/ask?tags=workbox
    🐞 Found a bug? Report it on GitHub
       https://github.com/GoogleChrome/workbox/issues/new
▼ workbox Router is responding to: /
    workbox Found a route to handle this request:
    ▶ RegExpRoute {handler: NetworkFirst, match: f, method: "GET"}
  ▶ workbox View request details here.
▼ workbox Using NetworkFirst to respond to  '/'
    workbox Got response from network.
  ▶ workbox View the final response here.
```

- *Improved webpack Plugin.* `workbox-webpack-plugin` integrates more closely with the webpack build process, allowing for a zero-config use case when you want to precache all the assets in the build pipeline.

Achieving these goals, and cleaning up some aspects of the previous interface that felt awkward or led to antipatterns, required introducing a number of breaking changes in the v3 release.

Breaking Changes

Build Configuration

The following changes affect the behavior of all of our build tools (`workbox-build`, `workbox-cli`, `workbox-webpack-plugin`), which share a common set of configuration options.

- The `'fastest'` handler name was previously valid, and treated as an alias for `'staleWhileRevalidate'`, when configuring `runtimeCaching`. It's no longer valid, and developers should switch to using `'staleWhileRevalidate'` directly.
- Several `runtimeCaching.options` property names have been updated, and additional parameter validation is in place that will cause a build to fail if an invalid configuration is used. See the documentation for `runtimeCaching` for a list of currently supported options.

workbox-background-sync

- The `maxRetentionTime` configuration parameter is now interpreted as a number of minutes, rather than a number of milliseconds.
- There is now a required string, representing the queue name, that must be passed in as the first parameter when constructing either the Plugin or standalone class. (It was previously passed in as a property of the options.) Consult the documentation for the updated API surface.

workbox-broadcast-cache-update

- There is now a required string, representing the channel name, that must be passed in as the first parameter when constructing either the Plugin or standalone class.

For example, in v2 you'd initialize the Plugin class as follows:

```
new workbox.broadcastCacheUpdate.BroadcastCacheUpdatePlugin({
  channelName: 'cache-updates',
  headersToCheck: ['etag']
});
```

The equivalent usage in v3 is:

```
new workbox.broadcastUpdate.Plugin(
  'cache-updates',
  {headersToCheck: ['etag']}
);
```

Consult the documentation for the updated API surface.

workbox-build

- By default, `glob` pattern matching will now be performed with the options `follow: true` (which will follow symlinks) and `strict: true` (which is less tolerant of "unusual" errors). You can disable either and return to the previous behavior by setting `globFollow: false` and/or `globStrict: false` in your build configuration.
- The functions in `workbox-build` all return an additional property, `warnings`, in the responses that they return. Some scenarios that were treated as fatal errors in v2 are now allowed, but reported via `warnings`, which is an array of strings.

In v2, you might call `generateSW` like:

```
const workboxBuild = require('workbox-build');

workboxBuild.generateSW({...})
  .then(({count, size}) => console.log(`Precached ${count} files, totalling ${siz
  .catch((error) => console.error(`Something went wrong: ${error}`);
```

While you can use the same code in v3, it's a good idea to check for any `warnings` and log them:

```
const workboxBuild = require('workbox-build');

workboxBuild.generateSW({...})
  .then(({count, size, warnings}) => {
    for (const warning of warnings) {
      console.warn(warning);
    }
    console.log(`Precached ${count} files, totalling ${size} bytes.`);
  })
  .catch((error) => console.error(`Something went wrong: ${error}`);
```

- Developers who wrote their own custom `ManifestTransform` functions in v2 need to return the manifest array in a object (i.e. instead of `return manifestArray;` you should use `return {manifest: manifestArray};`).mThis allows your plugin to include an optional `warnings` property, which should be an array of strings containing non-fatal warning information.

If you were writing a custom `ManifestTransform` in v2, then code like:

```
const cdnTransform = (manifestEntries) => {
  return manifestEntries.map(entry => {
    const cdnOrigin = 'https://example.com';
    if (entry.url.startsWith('/assets/')) {
      entry.url = cdnOrigin + entry.url;
    }
    return entry;
  });
};
```

has a v3 equivalent of:

```
const cdnTransform = (manifestEntries) => {
  const manifest = manifestEntries.map(entry => {
    const cdnOrigin = 'https://example.com';
    if (entry.url.startsWith('/assets/')) {
      entry.url = cdnOrigin + entry.url;
```

```
    }
    return entry;
  });
  return {manifest, warnings: []};
};
```

- The `getFileManifestEntries()` function has been renamed to `getManifest()`, and the promise returned now includes additional information about the URLs which are precached.

Code like the following in v2:

```
const manifestEntries = await workboxBuild.getFileManifestEntries({...});
```

can be rewritten in v3 as:

```
const {manifestEntries, count, size, warnings} = await workboxBuild.getMani

// Use manifestEntries like before.
// Optionally, log the new info returned in count, size, warnings.
```

- The `generateFileManifest()` function has been removed. Developers are encouraged to call `getManifest()` instead, and use its response to write data to disk in the appropriate format.

workbox-cache-expiration

- The plugin API has stayed the same, which is the mode that most developers will end up using. However there are significant API changes impacting developers who use it as a standalone class. Consult the documentation for the updated API surface.

workbox-cli

Developers can run the CLI with the `--help` flag for a full set of supported parameters.

- Support for the `workbox-cli` alias for the binary script has been removed. The binary can now only be accessed as `workbox`.
- The v2 commands `generate:sw` and `inject:manifest` have been renamed to `generateSW` and `injectManifest` in v3.
- In v2, the default configuration file (used when one wasn't explicitly provided) was assumed to be `workbox-cli-config.js` in the current directory. In v3, it's `workbox-`

`config.js`.

Taken together, this means that in v2:

```
$ workbox inject:manifest
```

would run the "inject manifest" build process, using a configuration read from `workbox-cli-config.js`, and in v3:

```
$ workbox injectManifest
```

will do the same, but read the configuration from `workbox-config.js`.

workbox-precaching

- The `precache()` method previously performed both the cache modifications and set up routing to serve cached entries. Now, `precache()` only modifies cache entries, and a new method, `addRoute()`, has been exposed to register a route to serve those cached responses. Developers who want the previous, two-in-one functionality can switch to calling `precacheAndRoute()`.
- Several options which used to be configured via the `WorkboxSW` constructor are now passed in as the `options` parameter in `workbox.precaching.precacheAndRoute([...], options)`. The defaults for those options, when not configured, are listed in the reference docs.
- By default, URLs that lack any file extension will automatically be checked for a match against a cache entry containing a `.html` extension. For instance, if a request is made for `/path/to/index` (which isn't precached) and there's a precached entry for `/path/to/index.html`, that precached entry will be used. Developers can disable this new behavior by setting `{cleanUrls: false}` when passing options into `workbox.precaching.precacheAndRoute()`.
- `workbox-broadcast-update` will no longer be automatically configured to announce cache updates for precached assets.

The following code in v2:

```
const workboxSW = new self.WorkboxSW({
  directoryIndex: 'index.html',
  ignoreUrlParametersMatching: [/^utm_/],
  precacheChannelName: 'precache-updates',
});
workboxSW.precache([...]);
```

has a v3 equivalent of:

```
workbox.precaching.addPlugins([
    new workbox.broadcastUpdate.Plugin('precache-updates')
]);

workbox.precaching.precacheAndRoute([...], {
  cleanUrls: false,
  directoryIndex: 'index.html',
  ignoreUrlParametersMatching: [/^utm_/],
});
```

workbox-routing

- Developers who previously used `workbox-routing` via a WorkboxSW object's `workbox.router.*` namespace need to switch to the new namespace, `workbox.routing.*`.
- Routes are now evaluated in a first-registered-wins order. This is the **opposite** order of `Route` evaluation that was used in v2, where the last-registered `Route` would be given precedence.
- The `ExpressRoute` class, and **support for "Express-style" wildcards have been removed**. This reduces the size of `workbox-routing` considerably. Strings used as the first parameter to `workbox.routing.registerRoute()` will now be treated as exact matches. Wildcard or partial matches should be handled by `RegExp`s—using any `RegExp` that matches against part or all of the request URL can trigger a route.
- The `addFetchListener()` helper method of the `Router` class has been removed. Developers can either add their own `fetch` handler explicitly, or use the interface provided by `workbox.routing`, which will implicitly create a `fetch` handler for them.
- The `registerRoutes()` and `unregisterRoutes()` methods were removed. The versions of those methods that operate on a single `Route` were not changed, and developers who need to register or unregister multiple routes at once should make a series of calls to `registerRoute()` or `unregisterRoute()` instead.

The following code in v2:

```
const workboxSW = new self.WorkboxSW();

workboxSW.router.registerRoute(
  '/path/with/.*/wildcard/',
  workboxSW.strategies.staleWhileRevalidate(),
);
```

```
workboxSW.router.registerRoute(
  new RegExp('^https://example.com/'),
  workboxSW.strategies.networkFirst(),
);
```

has a v3 equivalent of:

```
workbox.routing.registerRoute(
  new RegExp('^https://example.com/'),
  workbox.strategies.networkFirst(),
);

workbox.routing.registerRoute(
  new RegExp('^/path/with/\.*/wildcard'),
  workbox.strategies.staleWhileRevalidate()
);
```

workbox-strategies (formerly know as workbox-runtime-caching)

- The **`workbox-runtime-caching`** module is now officially known as **`workbox-strategies`**, and has been published on **`npm`** under its new name.
- Using cache expiration in a strategy without also supplying a cache name is no longer valid. In v2, this was possible:

```
workboxSW.strategies.staleWhileRevalidate({
  cacheExpiration: {maxEntries: 50},
});
```

This would lead to expiring entries in the default cache, which is unexpected. In v3, a cache name is required:

```
workboxSW.strategies.staleWhileRevalidate({
  cacheName: 'my-cache',
  plugins: [
    new workbox.expiration.Plugin({maxEntries: 50})
  ]
});
```

- The **`cacheWillMatch`** lifecycle method has been renamed to **`cachedResponseWillBeUsed`**. This should not be a visible change for developers unless they wrote their own plugins that reacted to**`cacheWillMatch`**.

- The syntax for specifying plugins when configuring a strategy has changed. Each plugin needs to be explicitly listed in the **`plugins`** property of the strategy's configuration.

The following code in v2:

```
const workboxSW = new self.WorkboxSW();

const networkFirstStrategy = workboxSW.strategies.networkFirst({
  cacheName: 'my-cache',
  networkTimeoutSeconds: 5,
  cacheExpiration: {
    maxEntries: 50,
  },
  cacheableResponse: {
    statuses: [0, 200],
  }
});
```

has a v3 equivalent of:

```
const networkFirstStrategy = workbox.strategies.networkFirst({
  cacheName: 'my-cache',
  networkTimeoutSeconds: 5,
  plugins: [
    new workbox.expiration.Plugin({maxEntries: 50}),
    new workbox.cacheableResponse.Plugin({statuses: [0, 200]}),
  ],
});
```

You can learn more in the "Using Plugins" guide.

workbox-sw

- Under the hood, **`workbox-sw`** has been rewritten to be a lightweight "loader" interface, that takes some basic configuration and is responsible for pulling in the other modules that are needed at runtime. Instead of constructing a new instance of the **`WorkboxSW`** class, developers will interact with an existing instance that's automatically exposed in the global namespace.

Previously in v2:

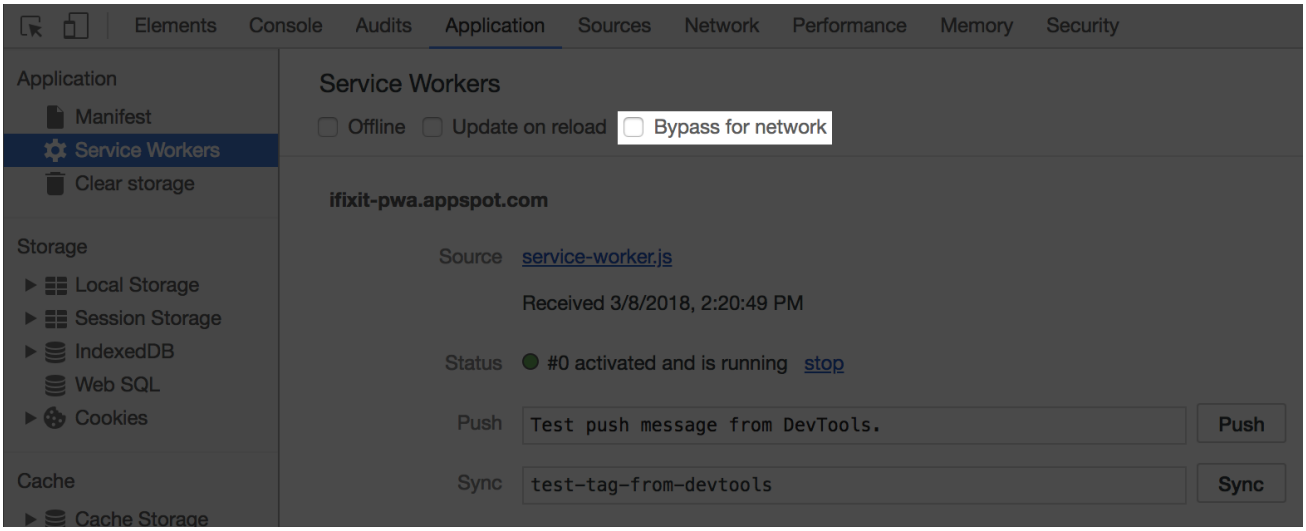
```
importScripts('<path to workbox-sw>/importScripts/workbox-sw.prod.v2.1.3.js
```

```
const workbox = new WorkboxSW({
  skipWaiting: true,
  clientsClaim: true,
  // etc.
});

workbox.router.registerRoute(...);
```

In v3, you just have to import the `workbox-sw.js` script, and a ready-to-use instance will be automatically available in the global namespace as `workbox`:

```
importScripts('<path to workbox-sw>/3.0.0/workbox-sw.js');

// workbox is implicitly created and ready for use.
workbox.routing.registerRoute(...);
```

- `skipWaiting` and `clientsClaim` are no longer options passed to the `WorkboxSW` constructor. Instead, they have been changed to the methods `workbox.clientsClaim()` and `workbox.skipWaiting()`.
- The `handleFetch` option that was previously supported in the v2 constructor is no longer supported in v3. Developers who need similar functionality to test their service worker without any fetch handlers being invoked can use the "Bypass for network" option available in Chrome's Developer Tools.



workbox-webpack-plugin

The plugin has been substantially rewritten, and in many cases, can be used in a "zero-configuration" mode. Consult the documentation for the updated API surface.

- The API now exposes two classes, `GenerateSW` and `InjectManifest`. This makes the toggling between modes explicit, vs. the v2 behavior where behavior changed based on the presence of `swSrc`.
- By default, assets in the webpack compilation pipeline will be precached, and it is no longer necessary to configure `globPatterns`. The only reason to continue using `globPatterns` is if you need to precache assets that are not included in your webpack build. In general, when migrating to the v3 plugin, you should start by removing all of your previous `glob`-based configuration, and only re-add it if you specifically need it.

Getting help

We anticipate most migrations to be straightforward. If you run into issues not covered in this guide, please let us know by opening an issue on GitHub.

Last updated April 17, 2018.