Migrate from sw-precache or sw-toolbox

Developers who have previously used **sw-precache** and/or **sw-toolbox** have a straightforward upgrade path to the Workbox family of libraries. Upgrading to Workbox will provide a modern, extensible service worker experience with improved debugging and developer ergonomics.

Modifications to your existing configuration

If you're using **sw-precache** configured with any of the following options, you'll need to take the following changes into account when migrating to Workbox.

Renamed options

The **dynamicUrlToDependencies** config parameter has been renamed **templatedUrls**.

The **staticFileGlobs** config parameter has been renamed **globPatterns**.

The **runtimeCaching** config parameter takes an updated set of options, corresponding to the names used in the underlying Workbox modules. To illustrate what's been renamed, this **sw-precache** configuration:

```
runtimeCaching: [{
  urlPattern: /api/,
  handler: 'fastest',
  options: {
    cache: {
      name: 'my-api-cache',
      maxEntries: 5,
      maxAgeSeconds: 60,
    },
  },
}],
```

is equivalent to this Workbox configuration:

```
runtimeCaching: [{
  urlPattern: /api/,
  // 'fastest' is now 'staleWhileRevalidate'
  handler: 'staleWhileRevalidate',
  options: {
    // options.cache.name is now options.cacheName
```

```
    cacheName: 'my-api-cache',
    // options.cache is now options.expiration
    expiration: {
      maxEntries: 5,
      maxAgeSeconds: 60,
    },
  },
}],
```

Deprecated options

Express-style wildcard routes are no longer supported. If you were using Express-style wildcard routes in either the **`runtimeCaching`** configuration or directly in **`sw-toolbox`**, please migrate to an equivalent regular expression route when using Workbox.

sw-precache migrations

From the sw-precache CLI to workbox-cli

Developers using the **`sw-precache`** command line interface, either running the command manually or as part of an **`npm scripts`**-based build process, will find using the **`workbox-cli`** module to be the easiest way to migrate. Installing **`workbox-cli`** will give you access to a binary called **`workbox`**.

While the **`sw-precache`** CLI supported configuring via either command line flags or a configuration file, the **`workbox`** CLI requires that all configuration options be provided in a configuration file, using CommonJS **`module.exports`**.

The **`workbox`** CLI supports a number of different modes. (Use **`workbox --help`** to see all of them.) But the mode that most closely matches **`sw-precache`**'s functionality is **`generateSW`**. So a call to

```
$ sw-precache --config='sw-precache-config.js'
```

can be expressed as

```
$ workbox generateSW workbox-config.js
```

From the sw-precache node module to the workbox-build node module

Developers using the **node** API for **sw-precache**, either as part of a **gulp**/**Grunt** workflow or just within a custom **node** build script, can migrate by switching to the **workbox-build** **node** module.

The **workbox-build** module's **generateSW()** function most closely matches the **sw-precache** module's **write()** function. One key difference is that **generateSW()** always returns a Promise, while the old **write()** function supported both a callback and a Promise-based interface.

gulp usage along the lines of

```
const swPrecache = require('sw-precache');
gulp.task('generate-service-worker', function() {
  return swPrecache.write('service-worker.js', {
    // Config options.
  });
});
```

can be changed to

```
const workboxBuild = require('workbox-build');
gulp.task('generate-service-worker', function() {
  return workboxBuild.generateSW({
    // Config options.
  });
});
```

From the sw-precache-webpack-plugin to the Workbox webpack plugin

Developers using the **sw-precache-webpack-plugin** as part of their webpack build process can migrate by switching to the **GenerateSW** class within the **workbox-webpack-plugin** module.

workbox-webpack-plugin integrates directly with the webpack build process and "knows" about all of the assets generated by a given compilation. This means that, for many use cases, you can rely on the default behavior of **workbox-webpack-plugin** without additional configuration, and get an equivalent service worker to what **sw-precache-webpack-plugin** provides.

```
const SWPrecacheWebpackPlugin = require('sw-precache-webpack-plugin');
const webpackConfig = {
  // ...
  plugins: [
```

```
    new SWPrecacheWebpackPlugin({
      dontCacheBustUrlsMatching: /\.\w{8}\./,
      filename: 'service-worker.js',
    }),
  ],
};
```

can be changed to

```
const {GenerateSW} = require('workbox-webpack-plugin');
const webpackConfig = {
  // ...
  plugins: [
    new GenerateSW({
      // Config options, if needed.
    }),
  ],
};
```

sw-toolbox migrations

Migrate from hand-crafted sw-toolbox to workbox-sw

If you were using **`sw-toolbox`** directly (rather than using it implicitly via **`sw-precache`**'s **`runtimeCaching`** option), then the migration to Workbox requires some manual adjustments to get the equivalent behavior. For more context, read the documentation for the **`workbox-routing`** and **`workbox-strategies`** modules can help provide more context.

Here are some code snippets to help guide the migration. This **`sw-toolbox`** code:

```
importScripts('path/to/sw-toolbox.js');

// Set up a route that matches HTTP 'GET' requests.
toolbox.router.get(
  // Match any URL that contains 'ytimg.com', regardless of
  // where in the URL that match occurs.
  /\.ytimg\.com\//,

  // Apply a cache-first strategy to anything that matches.
  toolbox.cacheFirst,

  {
    // Configure a custom cache name and expiration policy.
    cache: {
```

```
      name: 'youtube-thumbnails',
      maxEntries: 10,
      maxAgeSeconds: 30
    }
  }
);

// Set a default network-first strategy to use when
// there is no explicit matching route:
toolbox.router.default = toolbox.networkFirst;
```

is equivalent to this Workbox code:

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

workbox.router.registerRoute(
  // Match any URL that contains 'ytimg.com'.
  // Unlike in sw-toolbox, in Workbox, a RegExp that matches
  // a cross-origin URL needs to include the initial 'https://'
  // as part of the match.
  new RegExp('^https://.*\.ytimg\.com'),

  // Apply a cache-first strategy to anything that matches.
  workbox.strategies.cacheFirst({
    // Configuration options are passed in to the strategy.
    cacheName: 'youtube-thumbnails',
    plugins: [
      new workbox.expiration.Plugin({
        maxEntries: 10,
        maxAgeSeconds: 30,
      }),
      // In Workbox, you must explicitly opt-in to caching
      // responses with a status of 0 when using the
      // cache-first strategy.
      new workbox.cacheableResponse.Plugin({
        statuses: [0, 200],
      }),
    ]
  })
);

// Set a default network-first strategy to use when
// there is no explicit matching route:
workbox.router.setDefaultHandler(workbox.strategies.networkFirst());
```

Getting help

We anticipate most migrations to Workbox 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 March 14, 2018.