EcomLabs

STEP-BY-STEP GUIDE

hreflang in Next.js for a bilingual site

The helper that generates reciprocal hreflang tags in Next.js, the sitemap with the language versions, which code to use (es or es-MX) and a Node script that validates your pages' HTML. With a checklist.

FOR	Next.js (App Router) sites with an English and a Spanish version
TESTED WITH	Node 22.22.3 · Next.js 15 metadata resolvers and tsc --strict · public HTML saved on November 12, 2026
UPDATED	November 12, 2026
DOCS	developers.google.com/search/docs/specialty/international/localized-versions
ARTICLE	ecomlabs.dev/g/hreflang-en
AUTHOR	Luis Morales · @luismoralesdev

Contents

Before you start	Google's rules
Step 01	The helper and the page
Step 02	The sitemap
Step 03	es or es-MX
Step 04	The validation script
When you finish	Checklist

BEFORE YOU START

Google's rules

Per Google Search Central, these are the rules that make hreflang work:

- Each version lists itself and all the others. If two pages do not point to each other, the tags are ignored.
- URLs are fully qualified, with `https://`.
- The code is the language (ISO 639-1) and, optionally, the region (ISO 3166-1 alpha-2): `es` and `es-MX` yes; a bare region and `es-419` no.
- `x-default` is the value for when no other version matches.
- HTML, HTTP header and sitemap are equivalent; using all three at once gives no benefit in Search.

STEP 01

The helper and the page

One function builds the list, so both versions use the same one and reciprocity holds by construction. Checked with `tsc --strict` against the Next.js 15 types:

```
// lib/hreflang.ts
import type { Metadata } from "next";

export type Locale = "en" | "es";

// English lives at the root and Spanish under /es. Each version's whole path is
// passed in, because the slug can differ between languages.
export function pageAlternates(paths: Record<Locale, string>, current: Locale): Metadata["alternates"] {
  return {
    canonical: paths[current],
    languages: { en: paths.en, es: paths.es, "x-default": paths.en },
  };
}

// app/blog/[slug]/page.tsx (the English page; the Spanish page passes "es")
export async function generateMetadata({ params }: Props): Promise<Metadata> {
  const { slug } = await params;
  const post = await getPost(slug); // { slug: { en, es } } from your data layer
  return {
    title: post.title,
    alternates: pageAlternates(
      { en: `/blog/${post.slug.en}`, es: `/es/blog/${post.slug.es}` },
      "en",
    ),
  };
}
```

```
// app/layout.tsx: lets the relative paths above resolve to full URLs
export const metadata: Metadata = { metadataBase: new URL("https://example.com") };
```

Replace `https://example.com` with your domain. Without `metadataBase`, a relative path breaks the build. Static pages (home, services) use the same helper with their two paths.

STEP 02

The sitemap

Since Next.js 14.2.0, each sitemap entry accepts `alternates.languages`. The example in the documentation does not list the entry's own version; Google asks for it, so it is here:

```
// app/sitemap.ts
import type { MetadataRoute } from "next";

const BASE = "https://example.com";

// One entry per page, with the path of each language version.
const pages = [
  { en: "/", es: "/es" },
  { en: "/services", es: "/es/servicios" },
];

export default function sitemap(): MetadataRoute.Sitemap {
  return pages.flatMap((p) => {
    const languages = { en: BASE + p.en, es: BASE + p.es, "x-default": BASE + p.en };
    // Both versions get a <url>, and both carry the same full list.
    return [
      { url: languages.en, alternates: { languages } },
      { url: languages.es, alternates: { languages } },
    ];
  });
}
```

STEP 03

es or es-MX

CODE	SUPPORTED?	WHEN
`es`	Yes	All Spanish, no region
`es-MX`	Yes	A version specific to Mexico
`es-419`	No	Not ISO 639-1 plus ISO 3166-1
`MX`	No	A region cannot stand alone
`x-default`	Yes (reserved)	When no other version matches

If you have a Mexico version and another for the rest, list `es-MX` and `es`, like Google's example with `en-GB`, `en-US` and `en`. Next.js types accept `es-419` without complaint: the error shows up in Google, not in your editor.

STEP 04

The validation script

Save `check-hreflang.mjs`. It reads saved HTML, with no dependencies:

```
// Usage: node check-hreflang.mjs page-en.html page-es.html [more.html]
// Each file is the saved HTML of one URL. The pages you pass form the set
// that gets checked for return links.
import { readFileSync } from "node:fs";

const CODE = /^(x-default|[a-z]{2}(-[A-Z]{2})?)$/; // ISO 639-1, optional ISO 3166-1 alpha-2
const attr = (tag, name) => tag.match(new RegExp(`\\s${name}\\s*=\\s*"([^"]*)"`, "i"))?.[1];
const norm = (u) => (u ?? "").replace(/\/$/, "");

function parse(file) {
  const tags = readFileSync(file, "utf8").match(/<link\b[^>]*>/gi) ?? [];
  const canonical = tags.filter((t) => attr(t, "rel") === "canonical").map((t) => attr(t, "href"))[0];
  const alternates = tags
    .filter((t) => attr(t, "rel") === "alternate" && attr(t, "hreflang"))
    .map((t) => ({ lang: attr(t, "hreflang"), href: attr(t, "href") }));
  return { file, canonical, alternates };
}

const pages = process.argv.slice(2).map(parse);
const byUrl = new Map(pages.map((p) => [norm(p.canonical), p]));
let failed = false;

for (const p of pages) {
  const errors = [];
  const warnings = [];
  if (!p.canonical) errors.push("no rel=canonical (used here as the page's own URL)");
```

```
  if (p.alternates.length === 0) errors.push("no hreflang tags");
  for (const { lang, href } of p.alternates) {
    if (!CODE.test(lang)) errors.push(`unsupported code "${lang}"`);
    if (!/^https?:\/\//.test(href ?? "")) errors.push(`href is not fully qualified: "${href}"`);
  }
  if (p.alternates.length && !p.alternates.some((a) => a.lang === "x-default")) warnings.push("no x-default");
  if (p.alternates.length && !p.alternates.some((a) => norm(a.href) === norm(p.canonical))) {
    errors.push("the page does not list itself");
  }
  for (const { lang, href } of p.alternates) {
    const other = byUrl.get(norm(href));
    if (other && other !== p && !other.alternates.some((a) => norm(a.href) === norm(p.canonical))) {
      errors.push(`${lang} -> ${href}: that page does not link back`);
    }
  }
  const lines = [...errors.map((e) => `\n     - ${e}`), ...warnings.map((w) => `\n     ! ${w}`)];
  console.log(`${errors.length ? "FAIL" : "OK  "} ${p.file}${lines.join("")}`);
  if (errors.length) failed = true;
}
process.exit(failed ? 1 : 0);
```

Save the HTML of each pair with `curl -sS -o home-en.html https://your-domain.com/` and `curl -sS -o home-es.html https://your-domain.com/es`, and pass both files together: `node check-hreflang.mjs home-en.html home-es.html`. It exits with code 1 on errors. A missing `x-default` warns, it does not fail. Check one pair of each page type: home, product, article.

> **CAUTION**
>
> The script reads the HTML you give it, not the sitemap. Check the sitemap separately: each `<url>` must carry one `xhtml:link` per version, its own included. We tested it on saved public HTML of two page pairs of a real site (they pass) and on a broken page written by hand (it fails with four findings).

WHEN YOU FINISH

Checklist

- [ ] Every page lists all its versions and itself, with full URLs.
- [ ] The list comes from one function, and both versions use the same one.
- [ ] Each language has its own `canonical`.
- [ ] `x-default` points to the default version.
- [ ] You use `es`, or `es-MX` only if there is a Mexico version.
- [ ] Each language's slugs are stored, not computed with a prefix.
- [ ] `metadataBase` is set in the root layout and the sitemap carries the full list.
- [ ] The script passes on one page of each type.