Title: Using NPM for HATPro Development

|  |  |
| --- | --- |
| Document ID: | **DocID001** |
| Version: | **1.0** |
| Last Saved: | **0000-00-00 0:00:00** |
| Document Status: | **Draft** |
| Author: | **Neil Thomson** |
| Product/Release |  |

|  |  |
| --- | --- |
| Abstract:  (Comments) | <Enter the abstract in Properties/Summary/Comments> |

Detailed History of Changes

|  |  |  |  |
| --- | --- | --- | --- |
| Ver. | Date | Who | Detailed description of Changes |
| 1.0 | 08Dec2004 | NJT | Created |

Table of Contents

1. Introduction 1

2. First principles (what’s what) 1

3. Dependencies: dependencies vs devDependencies 1

4. What the common npm commands do 1

5. When do you run npm i -D chokidar-cli? 2

6. Concrete, step-by-step on Windows 2

6.1 One-time project setup (or on a new machine / fresh clone) 2

6.2 Daily workflow 3

6.3 After a reboot 3

6.4 After pulling changes that modify dependencies 3

7. Quick glossary 4

# Introduction

For those new to using NPM and specifically for the HATPro project using GitHub as the repository, the following provides guidance on how to set up on a new machine.

This was writen for Windows 11 development, but should apply to Apple/Mac and Linux/UNIX environments. That is one of the key reasons to use NPM to be the CLI for running the various HATPro tools to do a lint check on .PUML files, generate JSON Schema and Enum files.

# First principles (what’s what)

* **Node.js** = the JavaScript runtime you install on your computer.
* **npm** = “Node Package Manager.” It **comes with Node**. You don’t install npm with npm install; you install **Node**, and npm is included.
* Your project has a **package.json** file (project manifest) listing scripts and dependencies.
* When you “install a package,” npm puts files in a local **node\_modules/** folder and (optionally) records them in package.json and package-lock.json.

# Dependencies: dependencies vs devDependencies

* **dependencies**: needed to **run** your program in production.
* **devDependencies**: tools for **development** only (linters, CLIs, watchers, test frameworks).
* The flag **-D** (or --save-dev) tells npm: “record this in **devDependencies**.”

So:

npm i -D chokidar-cli

means:  
“Install the **chokidar-cli** tool for development and **add it to devDependencies** in package.json.”  
This is a **one-time** action (or any time you add/change tools), **not** something you do at every reboot.

# What the common npm commands do

* **npm install** (alias npm i):  
  Looks at package.json (+ package-lock.json if present) and installs everything listed into node\_modules/.  
  Use it when: first time on a machine, after cloning, or after dependencies change.
* **npm ci**:  
  Like npm install, but **clean and deterministic**: it **deletes** node\_modules/ and installs **exactly** what’s in package-lock.json—no version drift.  
  Use it when: you have a package-lock.json and want a guaranteed clean setup (recommended for CI and for you if the lockfile exists).
* **npm run <scriptName>**:  
  Runs a script defined in "scripts" inside package.json.  
  Example: npm run gen:schemas executes the command string assigned to that name.
* **npx <tool>**:  
  Runs a tool **without** adding it globally—prefers the one installed in your project.  
  Example: npx chokidar ... will run the locally installed chokidar-cli.

# When do you run npm i -D chokidar-cli?

Only when you decide you want “watcher” behavior (auto-regenerate on file save). It:

1. Adds "chokidar-cli": "x.y.z" to devDependencies in your **package.json** (persisted to disk).
2. Adds the files to **node\_modules/** so your watch scripts can execute it.

You **do not** need to repeat this every reboot. You only need to reinstall after:

* you **clone** the repo on a new machine, or
* you **deleted** node\_modules/, or
* package.json/package-lock.json changed and you need updated packages.

In those cases, just run **npm ci** (or npm install), and it will restore **everything**, including chokidar-cli.

# Concrete, step-by-step on Windows

## One-time project setup (or on a new machine / fresh clone)

1. Install **Node.js** (which includes npm). Check:
2. node -v
3. npm -v
4. In your repo root (where package.json lives):
   * If package-lock.json exists:
   * npm ci
   * Otherwise:
   * npm install
5. (Optional) If you want auto-watch:
6. npm i -D chokidar-cli

This updates package.json → devDependencies. Commit the changes if this repo is shared.

1. Add watch scripts (only if you want the auto-watch behavior) **to your package.json**:
2. {
3. "scripts": {
4. "watch": "chokidar \"packages/\*\*/puml/\*\*/\*.puml\" -c \"npm run gen:enums && npm run gen:schemas\"",
5. "watch:enums": "chokidar \"packages/\*\*/puml/\*\*/\*.puml\" -c \"npm run gen:enums\"",
6. "watch:schemas": "chokidar \"packages/\*\*/puml/\*\*/\*.puml\" -c \"npm run gen:schemas\""
7. }
8. }

(These are **source** changes; they live in package.json permanently.)

## Daily workflow

1. Open **Windows Terminal** (PowerShell tab) in your repo root.
2. Initialize today’s session (build once):
3. npm run build

(Runs: lint → gen:enums → gen:schemas → validate)

1. Open a **second** terminal in the repo root and start the watcher:
2. npm run watch

Leave it running. Every time you save a .puml, enums + schemas regenerate.

To stop watching: press **Ctrl+C** in that watcher terminal.

## After a reboot

* You **do not** need to reinstall chokidar or anything else unless you deleted node\_modules/.
* Just open the terminal, ensure you’re in the repo root, and run the scripts you need:
  + One-off build: npm run build
  + Watcher: npm run watch

## After pulling changes that modify dependencies

* If the team updated package.json or package-lock.json, do:
* npm ci

to sync your node\_modules/ with the lockfile.

# Quick glossary

* **package.json**: your project manifest (scripts + dependency list). You edit this when adding scripts or tools.
* **package-lock.json**: exact versions installed—used by npm ci to reproduce the same setup.
* **node\_modules/**: the installed packages. You don’t edit this; npm writes it.
* **devDependencies**: tools for development only (like **chokidar-cli**). They’re persisted to package.json and restored by npm ci / npm install.