GETTING STARTED

Workbench for Zephyr in VS Code

Installing the extension and blinking an LED on the Raspberry Pi Pico

A Practical, Step-by-Step Tutorial

Covers: Extension installation · Host tools · Zephyr SDK · West workspace · Blinky sample · Build & Flash
Prepared for Raspberry Pi Pico / Pico 2 development boards

1. Overview

Zephyr RTOS is a small, scalable, open-source real-time operating system built for resource-constrained and embedded devices. “Workbench for Zephyr” is a Visual Studio Code extension developed by Ac6 that brings the entire Zephyr workflow — host tool installation, SDK/toolchain management, West workspace setup, project creation, building, flashing, and debugging — directly into VS Code, removing most of the manual command-line setup that Zephyr traditionally requires.

This tutorial walks through installing Workbench for Zephyr in VS Code and using it to build and flash the classic “Blinky” sample application onto a Raspberry Pi Pico (RP2040) or Pico 2 (RP2350) board, so you can confirm your toolchain works end-to-end by watching an on-board LED blink.

What you will do

1. Install Visual Studio Code and the Workbench for Zephyr extension.
2. Install the required host tools (Python, CMake, Ninja, Git, and more).
3. Add a Zephyr SDK toolchain.
4. Create a West workspace using the built-in Raspberry Pi Pico template.
5. Create a new application from the Blinky sample and target the Pico board.
6. Build the firmware.
7. Flash it to the Raspberry Pi Pico and watch the LED blink.

Prerequisites

Requirement	Notes
Computer	Windows 10/11, Linux (x86_64), or macOS (Apple Silicon / aarch64)
Visual Studio Code	Recent version, installed from code.visualstudio.com
Internet connection	Needed to download the extension, host tools, SDK, and Zephyr sources
Raspberry Pi Pico or Pico 2	Any RP2040 or RP2350 based Pico board with a USB cable
macOS only	Homebrew must be installed beforehand (required for host tool installation)
Linux only	A Bash shell and your distribution’s package manager; sudo rights are needed to install some packages
Windows only	Nothing to pre-install — all tools are downloaded as portable versions automatically

> **Note**
> No prior Zephyr command-line experience is required. Workbench for Zephyr wraps the underlying “west” build tool with wizards and buttons, though it still uses the same Zephyr build system under the hood.

2. Step 1 — Install Visual Studio Code

1. Download Visual Studio Code from https://code.visualstudio.com for your operating system.
2. Run the installer and accept the default options.
3. Launch VS Code once to confirm it opens correctly.

3. Step 2 — Install the Workbench for Zephyr Extension

1. Open VS Code and click the Extensions icon in the Activity Bar on the left (or press Ctrl+Shift+X / Cmd+Shift+X).
2. In the search box, type "Workbench for Zephyr."
3. Select the extension published by Ac6 and click Install.

Installing Workbench for Zephyr automatically installs a set of companion extensions that it depends on:

- Devicetree Manager for Zephyr (Ac6) — visual editor for the devicetree and pin muxing
- Serial Monitor (Microsoft) — view the board's serial output
- C/C++ (Microsoft) — IntelliSense and debugging support for C/C++
- Cortex-Debug — debug backend for ARM Cortex-M targets
- File Downloader (Microsoft DevLabs) — used internally to download tools
- DeviceTree Language Server — language features for .dts / .overlay files

Once installed, a new Zephyr icon appears in the Activity Bar. Click it to open the "Workbench for Zephyr" panel, which is where the rest of this tutorial takes place.

4. Step 3 — Install Host Tools

Zephyr needs several host dependencies to build firmware, including CMake, Ninja, Python, a Device Tree Compiler, and Git. Workbench for Zephyr installs and manages these for you.

1. Open the Workbench for Zephyr tab in the Activity Bar.
2. Click Install Host Tools in the shortcuts view.
3. Wait for the installation to finish. It runs in an integrated terminal and can take several minutes; a progress notification is shown and can be cancelled at any time.

What gets installed, by platform

Platform	Behavior
Windows	Portable versions of Python, CMake, Ninja, gperf, Device Tree Compiler, Git, and wget are downloaded into a .zinstaller folder — no admin rights required.

Platform	Behavior
Linux	Python, CMake, and Ninja are downloaded directly. Remaining dependencies (git, gperf, dtc, gcc, make, ccache, dfu-util, and others) are installed using your distribution's package manager — you will be prompted for your sudo password.
macOS	Everything is installed through Homebrew: Python, CMake, Ninja, gperf, Device Tree Compiler, Git, plus supporting utilities.

> **Tip**
> If you already have some of these tools installed and want finer control — skipping tools, choosing which Python to use, or repairing a single tool — use "Advanced Host Tools Installation" instead of the one-click option.

After host tools finish installing, Workbench for Zephyr automatically installs the OpenOCD runner (used for flashing/debugging many boards) and offers to install additional runners for other probes. You can accept the default for now.

5. Step 4 — Add a Zephyr SDK Toolchain

The toolchain provides the cross-compiler used to build Zephyr firmware for the Pico's ARM Cortex-M processor.

1. In the Workbench for Zephyr panel, click Add Toolchain (or use the + button in the Toolchains view).
2. Under Toolchain family, keep the default selection: Zephyr SDK.
3. Set Source to Official.
4. Choose a Destination — Custom location (pick a short folder path such as D:\SDK or ~/zephyr-sdk) or Global (auto-discovered).
5. Choose the SDK Type: Full installs every architecture; Minimal lets you install only what you need.
6. If you chose Minimal, tick arm-zephyr-eabi — this is the toolchain used by the Pico's Cortex-M0+/M33 core.
7. Pick the latest Version from the list.
8. Click Import and wait for the download and installation to complete.

> **Note**
> Toolchains are registered globally in Workbench for Zephyr, so once installed, this SDK is available to any West workspace or application you create afterward.

6. Step 5 — Create a West Workspace for the Raspberry Pi Pico

A West workspace is the folder structure that holds the Zephyr source tree, its modules, and your applications. Workbench for Zephyr ships with a built-in template specifically for the Raspberry Pi Pico.

1. Click Add West Workspace in the shortcuts view (or the + button in the West Workspaces view).
2. Under Source location, keep From template.
3. Choose Minimal (fetches Zephyr plus only the modules needed for your board — faster than Full).
4. In Template, select Raspberry Pi Pico.
5. In Revision, keep the pre-selected latest release tag (or choose a specific Zephyr version).
6. Under Destination, click Browse… to choose a parent folder, and optionally rename the Subfolder (default is zephyrproject).
7. Leave the Advanced options at their defaults, unless you specifically want a dedicated Python virtual environment for this workspace.
8. Click Import.

A progress notification titled "Initializing west workspace" tracks manifest initialization, source download, and board discovery. This step downloads the Zephyr source tree and Pico-related modules, so it can take several minutes depending on your connection.

> **Tip**
> If a binary blob download fails during import, the workspace import still completes. You can retry later by right-clicking the workspace → Blobs → Blobs: Fetch.

7. Step 6 — Create the Blinky Application

With the workspace and toolchain ready, create a new application based on Zephyr's built-in Blinky sample.

1. Click Add Application in the shortcuts view (or the + button in the Applications view).
2. Select West Workspace: choose the Pico workspace you created in Step 5.
3. Select Toolchain: choose the Zephyr SDK you installed in Step 4.
4. Select Board: choose your board — rpi_pico for the original Pico (RP2040), or rpi_pico2 for the Pico 2 (RP2350). If it is not listed, choose "Enter custom board…" and type the board identifier.
5. Leave New or existing application on Create new application.
6. Select template: pick basic/blinky from the SAMPLES list.
7. Project Name: enter something like blinky_pico.
8. Application type: keep West workspace application (the project is created inside the workspace).
9. Leave Debug preset checked — it is useful while you are learning the toolchain.
10. Click Create.

Workbench for Zephyr generates the application folder and adds it to your VS Code workspace. A Zephyr application folder always contains, at minimum:

```
blinky_pico
  src/            // application source code (main.c)
  CMakeLists.txt  // links the app to Zephyr's CMake build system
  prj.conf        // Kconfig fragment (project configuration)
  build/          // build output (created after the first build)
```

> **Note**
> The Blinky sample already contains the C source that toggles a GPIO pin connected to the board's LED using Zephyr's GPIO driver API and the devicetree led0 alias — no code changes are required to blink the LED.

8. Step 7 — Build the Application

1. Open the Applications view in the Workbench for Zephyr panel.
2. Right-click on blinky_pico and choose Build — or click the gear/build icon next to the application.
3. Watch the build output in the panel at the bottom of VS Code.

A successful build compiles the Zephyr kernel and your application into a firmware image, and produces a .uf2 file in the build directory (build/zephyr/zephyr.uf2) — the Pico's standard format for drag-and-drop flashing — alongside the usual .elf and .hex outputs.

Tip

If the build fails with a toolchain or SDK version warning, it is usually non-blocking: Workbench for Zephyr will tell you which SDK version it recommends for your Zephyr revision. You can continue, or install the recommended SDK version from the Toolchains view.

9. Step 8 — Flash the Firmware to the Raspberry Pi Pico

The Pico ships with a UF2 bootloader: holding the BOOTSEL button while plugging it in mounts it as a USB mass-storage drive that accepts firmware by drag-and-drop. Workbench for Zephyr can drive this automatically through West, or you can flash manually.

Option A — Flash from Workbench for Zephyr

1. Press and hold the BOOTSEL button on the Pico, plug it into your computer via USB, then release BOOTSEL. The board mounts as a mass-storage drive (usually named RPI-RP2).
2. In the Applications view, right-click blinky_pico and choose Flash/Run — or click the Flash (run icon) button next to the application.
3. If no default flash runner is set yet, a "Select flash runner" picker opens. Choose the runner marked (compatible) — typically uf2 for the Pico.
4. Leave Custom Arguments For Runner empty and press Enter to flash with the defaults.

Behind the scenes this runs the following command in a terminal, using the board and build directory of your application:

```
west flash --board rpi_pico --build-dir blinky_pico/build
```

Option B — Manual drag-and-drop flashing

1. Put the Pico into BOOTSEL mode as described above.
2. In VS Code, right-click blinky_pico → Open Containing Folder, and browse into build/zephyr/.
3. Drag zephyr.uf2 onto the RPI-RP2 drive that appeared on your computer.
4. The Pico automatically reboots and disconnects the drive once the copy completes — the new firmware is now running.

> **Note**
> Set a default flash runner once (right-click the application → Build Configuration → Set Default Flash Runner) if you want to skip the runner-selection prompt on every future flash.

10. Step 9 — Verify the LED Is Blinking

Once flashing completes, the Pico restarts and runs the new firmware immediately — no further action is needed.

- Look at the small LED next to the USB connector on the Pico board.
- With the default Blinky sample, the LED should turn on and off roughly once per second.

- If nothing happens, unplug and replug the board (without holding BOOTSEL this time) to do a normal power cycle.

Congratulations — you have installed Workbench for Zephyr, set up a full Zephyr build environment, and built and flashed your first Zephyr application to a Raspberry Pi Pico.

11. Troubleshooting

Symptom	What to try
“Homebrew is not installed” error (macOS)	Install Homebrew from https://brew.sh, then re-run Install Host Tools.
Board not detected / no RPI-RP2 drive	Make sure you held BOOTSEL while plugging in the USB cable, and try a different USB cable/port (some cables are power-only).
SDK / Zephyr version mismatch warning	Non-blocking; open the Compatibility Matrix link in the warning and, if needed, install the recommended SDK version from the Toolchains view.
“PermissionError: Access is denied” while importing a workspace (Windows)	Close VS Code, run “west init -m <zephyr_repo_url> --mr <version> <dest_dir>” manually in a terminal, then import that folder using the Local folder source.
Build succeeds but flash fails	Confirm the selected flash runner is compatible with rpi_pico/rpi_pico2, and that the board is in BOOTSEL mode before flashing.
Blob download fails during workspace import	The workspace import still completes; right-click the workspace → Blobs → Blobs: Fetch to retry.

12. Next Steps

- Open src/main.c inside blinky_pico to see how the GPIO API and the led0 devicetree alias are used, then try changing the blink interval.
- Explore other samples under the SAMPLES list in the Add Application wizard — for example, samples involving buttons, sensors, or Bluetooth (on wireless-capable boards).
- Use the Serial Monitor extension to view printk() output from your application over USB.
- Try Debug from the Applications view to set breakpoints and step through code with Cortex-Debug.

Reference Documentation

- Workbench for Zephyr documentation: https://z-workbench.com/docs/documentation/zephyr-workbench
- Zephyr Project documentation: https://docs.zephyrproject.org
- Raspberry Pi Pico documentation: https://www.raspberrypi.com/documentation/microcontrollers/