01 — Getting Started

> **TraxProbe & Traxcope User Manual** · Version 1.0.0 · Updated 2026-08-24
 · https://embedya.com

This chapter takes you from an empty firmware project to live data on screen. It deliberately uses the smallest possible configuration; every step links to the reference chapter that covers it in depth.

What you will do:

1. Import TraxProbe — include directory, metadata linker script, and your `trax_config.h` folder
2. Fill in `trax_config.h` (hardware, timestamps, optional RTOS + version)
3. Initialize the probe and emit your first log message
4. Create a probe in Traxcope, connect, and stream

What you need:

- An embedded project (the walkthrough assumes an ARM Cortex-M MCU; Zynq, Xtensa/ESP32 and a Generic fallback port are also available)
- A transport to the PC — the simplest is **SEGGER RTT** via a J-Link debug probe (zero extra code); UART/TCP work too
- The Traxcope desktop application installed on your PC

Step 1 — Import TraxProbe

Copy (or submodule) the `TraxProbe` folder into your project tree, e.g. `Third_Party/TraxProbe/`.

TraxProbe is plain C with no build-system of its own. Whatever your environment — Eclipse/STM32CubeIDE, Keil, IAR, a Makefile, or CMake — the project needs **exactly three** settings. Nothing else.

What	Where	Why
Include directory	`Third_Party/TraxProbe/inc/`	All public headers. Hardware port, transport, and RTOS

What	Where	Why
		port resolve internally from the macros in your `trax_config.h`.
Metadata linker script	`Third_Party/TraxProbe/linker/trax_meta.ld`	Places names, format strings, units, and colors in dedicated ELF sections. Add it as an *additional* linker script (GNU ld: extra `-T`; Eclipse: the linker's script / "other flags" options). Your board's main linker script stays as it is.
`trax_config.h` folder	any folder you choose (this guide uses `App/Cfg/`)	The single user-editable config file. Add that folder to the include path so the library can `#include "trax_config.h"` by name.

Do not add extra include paths, and do not cherry-pick library source trees (`src/`, `transports/`, `os/`). Those are selected by the config macros in Step 2.

Without the linker fragment the metadata sections have no home and linking fails. ESP-IDF projects use the ldgen fragment `linker/esp/trax_meta.lf` instead — see Chapter 02.

Step 2 — Create `trax_config.h`

This is the **single user-editable configuration file**. It can live in any folder of your project — the only requirement is that the folder is on the include path (Step 1). This guide places it in `App/Cfg/`. Everything else in the library ships with working defaults. A minimal Cortex-M configuration using SysTick as the timestamp source:

```
#ifndef APP_TRAX_CONFIG_H_
#define APP_TRAX_CONFIG_H_

#include <stm32g0xx.h>   /* your device header, for SysTick below */

/* --- Hardware (MANDATORY) ---------------------------------------- */
#define TRAX_CFG_HW_PORT                TRAX_HW_PORT_ARM_CORTEX_M
```

```
/* --- Timestamps (MANDATORY) ----------------------------------------
 * Every trace event carries a timestamp. Here: SysTick down-counter
 * at 64 MHz, reloading every 1 ms (64000 cycles). */
#define TRAX_CFG_TIMESTAMP_TIMER_VAL   (SysTick->VAL)
#define TRAX_CFG_TIMESTAMP_TIMER_DIR   TRAX_TIMER_DIR_DOWN
#define TRAX_CFG_TIMER_FREQ_HZ         64000000U
#define TRAX_CFG_TICK_COUNTER_PERIOD   64000U

/* --- Transport (optional - RTT is the default) ---------------------- */
/* #define TRAX_CFG_TRANSPORT          TRAX_TRANSPORT_CUSTOM */

/* --- RTOS (optional - default is bare-metal) -----------------------
 * Selecting an RTOS also requires its version. The port uses the version
 * to pick the correct kernel hooks; omitting it is a compile error.
 * Read the triple from tskKERNEL_VERSION_NUMBER in FreeRTOS task.h. */
/* #define TRAX_CFG_RTOS_TYPE          TRAX_RTOS_FREERTOS */
/* #define TRAX_CFG_FREERTOS_VERSION   TRAX_FREERTOS_VERSION(11, 2, 0) */

#endif /* APP_TRAX_CONFIG_H_ */
```

That is the entire mandatory surface: **hardware port + timestamp description**. Transport defaults to RTT, RTOS defaults to none, buffers default to sensible sizes. Enable an RTOS from this same file when you need kernel tracing — and always pair `TRAX_CFG_RTOS_TYPE` with the matching version macro. The full mandatory/optional breakdown is in Chapter 02.

Step 3 — Hook the timestamp tick

In TICK_TIMER mode (the default) TraxProbe needs to know each time the tick timer wraps. Call `trax_timestamp_tick()` from your SysTick handler:

```
void SysTick_Handler(void)
{
    trax_timestamp_tick();
    /* ... your existing tick code ... */
}
```

Under FreeRTOS this happens automatically through the kernel trace hooks — no manual call needed (see Chapter 13).

Step 4 — Initialize and emit your first data

Bare-metal `main()`:

```c
#include <trax.h>

/* A log TID. For real projects put TIDs in small dedicated header
 * files — see Chapter 02. */
enum { LOG_TID_HELLO = TRAX_TID_RANGE_LOG_USER_START };

int main(void)
{
    bsp_init();                  /* clocks, SysTick, ... */

    trax_init();                 /* recording starts immediately */
    trax_wait_stream_started();  /* optional: block until Traxcope connects
*/

    uint32_t counter = 0;

    while (1) {
        TRAX_LOG_INFO(LOG_TID_HELLO, "demo", "Hello from the target!
count=%d", counter);
        counter++;

        delay_ms(100);

        trax_process();          /* drain ring buffer + handle host commands
*/
    }
}
```

Key points:

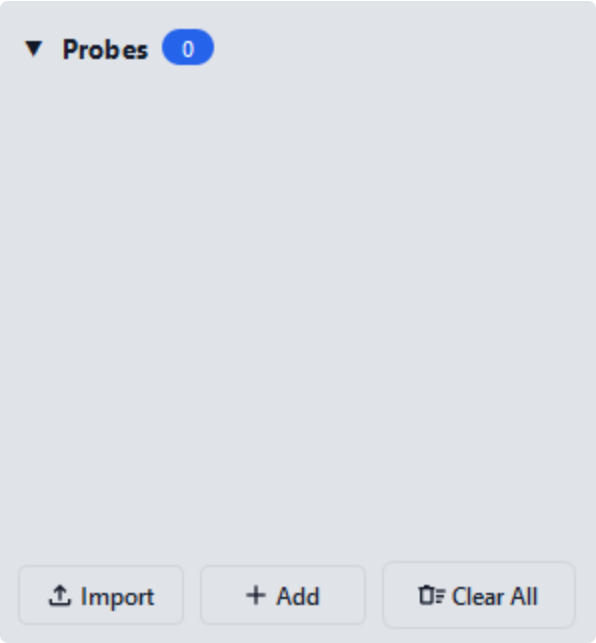
- `trax_init()` starts **recording** into the internal ring buffer immediately. Nothing is transmitted yet.
- `trax_process()` must be called periodically on bare metal — it transmits pending frames and processes host commands. Under FreeRTOS the library creates an internal **TraxCtrl** task that does this for you.
- **Accordion buffer.** The ring is a coherent recorder, not a best-effort FIFO. When producers outrun the transport (bandwidth leakage), streaming pauses, the ring drains, and a `STREAM_GAP` is placed between two coherent stream blocks. Everything before and after the gap stays valid and time-aligned; Traxcope draws the hole as an explicit gap, not a corrupted stream. Details in Chapter 02 §7.1.
- `trax_wait_stream_started()` is optional but recommended for demos: it blocks until Traxcope sends the start-streaming command, so you never lose startup events. Under FreeRTOS call it from a task, not from `main()` — see Chapter 02 §5.2.

- The format string in `TRAX_LOG_INFO` is **never** sent per-call at runtime. Only the TID and the raw argument values go on the wire for each log event. How Traxcope recovers the format string depends on `TRAX_CFG_META_STORAGE` in your `trax_config.h`:
  - `TRAX_META_IN_FLASH` **(default)** — metadata is compiled into flash and the firmware transmits the full schema to Traxcope once at session start. Traxcope does not need the `.elf` file.
  - `TRAX_META_ELF_ONLY` — metadata lives only in the `.elf`; nothing extra is sent at session start. Traxcope reads the format strings directly from the `.elf`, so the file path must be set in the probe configuration.

Build the project. If you use `TRAX_META_ELF_ONLY`, keep the resulting `.elf` file handy and point Traxcope to it in Step 5.

Step 5 — Create a probe in Traxcope

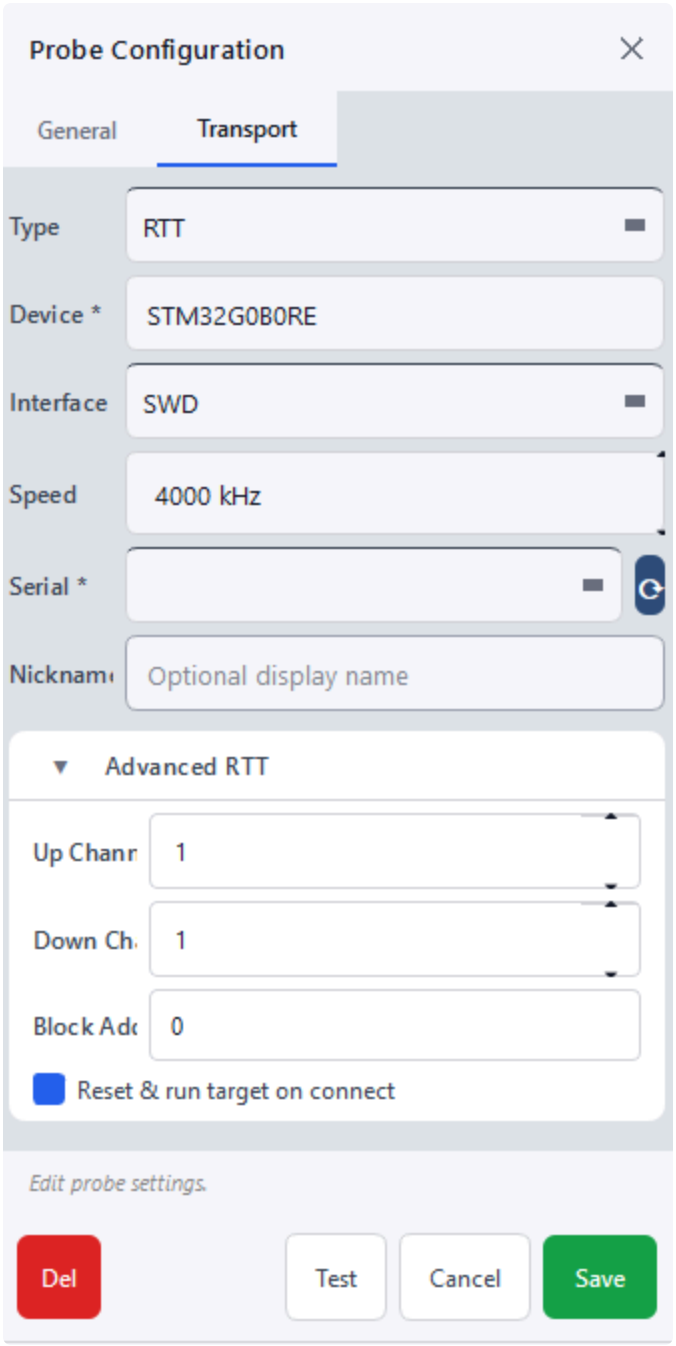
1. Start Traxcope.
2. In the left sidebar, open the **Probes** panel and click **Add**.



Screenshot: Sidebar Probes panel, Add button highlighted

3. In the probe configuration that opens:
   - **General tab** — give the probe a name. If you built with `TRAX_META_ELF_ONLY`, also set the **ELF file** path to the `.elf` you just built so Traxcope can read the format strings, names, units, and colors from it. With `TRAX_META_IN_FLASH` (the default) the ELF path is optional — the firmware streams the schema at session start.
   - **Transport tab** — select your transport:
     - **RTT** — select your J-Link (auto-detect available), choose interface/speed.
     - **UART** — select the COM port and baud rate matching your firmware transport.

    - **TCP** — enter IP address and port.
- Click **Test Connection** to verify the link before committing.



Screenshot: Probe config panel, Transport tab, RTT selected with J-Link detected

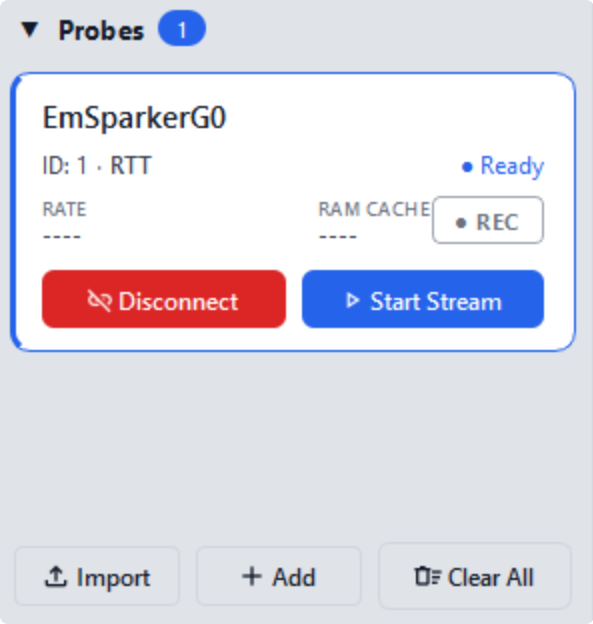
4. Save the probe. It appears as a card in the Probes panel.

Step 6 — Connect and stream

Each probe card has three controls (details in Chapter 03):

Button	What it does
Connect	Opens the transport and performs the session handshake

Button	What it does
Stream	Sends the start-streaming command — the firmware begins transmitting
Record	Persists the incoming stream to disk for later review

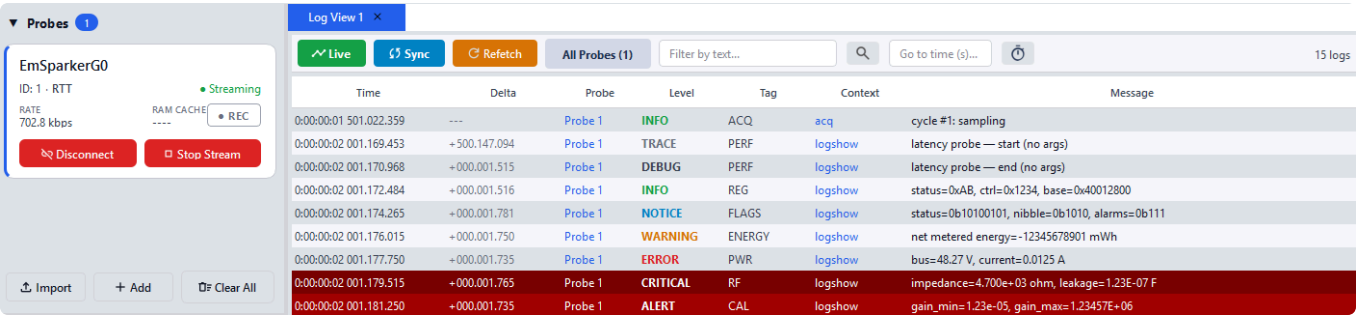


Screenshot: One probe card with Connect, Stream and Record buttons visible

Click **Connect**, then **Stream**. The firmware info card on the probe's General tab fills in with the project name, version, and build ID reported by the device.

Step 7 — Open views and see your data

Open the **Toolbox** panel in the sidebar and add a **Log** view. Your "Hello from the target!" lines appear, fully formatted with the incrementing count — 10 per second.



Screenshot: Main window, Log view with live data flowing

Congratulations — the pipeline works end to end. Plotting variables in the Scope view works the same way with `TRAX_VAR_SET` — see Chapter 11.

Where to go next

Goal	Chapter
Understand every config option, transports, ELF-only metadata, FreeRTOS	02 — TraxProbe Integration Guide
Master the Traxcope UI: recordings, workspaces, view sync	03 — Traxcope User Guide
Rich logging (levels, tags, hex dumps, log→var binding)	10 — Logging
Plot variables with units, formulas, math channels, cursors, power analysis	11 — Variables & Scope
High-rate sampled data and FFT analysis	12 — Stream Variables & Spectrum Analysis
See your FreeRTOS tasks, queues, and ISRs on a timeline	13 — RTOS Tracing
Measure code-section timing against deadlines	15 — Markers
Synchronize timelines across multiple boards	18 — Multi-Probe Synchronization