PicoMiteBT -- BLE Console for Pico 2 W

PicoMiteBT is a build variant of PicoMite MMBasic that replaces the USB CDC console with a Bluetooth Low Energy serial link. Instead of plugging a USB cable in for the BASIC prompt, you connect wirelessly from a phone, laptop, or desktop computer.

Internally it advertises a Nordic UART Service (NUS) over BLE -- the same GATT profile used by Adafruit Bluefruit, Nordic dev kits, and most other BLE-serial peripherals -- so the device works out of the box with any BLE-NUS-aware terminal app, and with a small Python bridge for OSes that don't have one (Windows, Linux).

What you need

- A Raspberry Pi Pico 2 W (the pico2_w board -- Pico 2 alone does not have a wireless chip).
- The PICOBTRP2350 firmware variant -- built by BuildPicoMite.bat alongside the standard set, or directly with CMake: cmake -DCOMPILE=PICOBTRP2350 -G "NMake Makefiles" ..
- A BLE-capable host: any modern phone, almost any Windows 10+ laptop, any Linux machine with BlueZ 5.x, any modern macOS.
- For desktop use: Python 3.8+ with bleak (pip install bleak).

The repo ships two scripts:

- ble_term.py -- recommended. A self-contained cross-platform VT100 terminal that talks directly to BLE NUS, with built-in XMODEM send/receive. No external terminal needed.
- ble_bridge.py -- alternative. BLE-to-TCP bridge for users who'd rather use their preferred terminal (Tera Term on Windows, PuTTY on Linux, etc.).

Device name

On startup the firmware builds a name from the last four hex characters of the chip's unique board ID, e.g. PicoMite-77BA. This name appears in every scan, and is what you point your tools at.

You can confirm the name on your specific board by scanning from a phone the first time.

Phone (Android / iOS)

Phones are the easiest path -- no bridge, no Python.

Android:

1. Install Serial Bluetooth Terminal (Kai Morich) from the Play Store.
2. Open the app, tap the menu -> Devices -> Bluetooth LE.
3. Tap the scan icon. Pick PicoMite-XXXX from the list.
4. The app will prompt you to pair -- accept (no PIN, "Just Works" pairing).
5. Back in the terminal view, you should see the MMBasic banner. Press Enter for the > prompt.

iOS:

1. Install Bluefruit Connect (Adafruit) -- free.
2. Connect to the PicoMite-XXXX device.

3. Use the UART tab. iOS handles the pairing transparently.

Subsequent re-connects don't require re-pairing -- the phone remembers the bond.

Desktop (Windows / Linux / macOS) -- ble_term

ble_term.py is the recommended desktop path: one script, no external terminal, works identically on all three OSes. It talks directly to the Pico's BLE NUS service, puts the host console into raw + virtual terminal mode, and forwards bytes both ways. The host terminal renders the VT100 escape sequences MMBasic emits natively.

Setup

1. Install bleak:

```
pip install bleak
```

(pip3 install bleak on macOS.)

2. Pair the device once through OS settings -- same as for the bridge path:

- Windows: Settings -> Bluetooth & devices -> Add device -> Bluetooth -> pick PicoMite-XXXX.
- Linux:

```
bluetoothctl
[bluetooth]# scan on
[bluetooth]# pair 28:CD:C1:FF:B7:8D
[bluetooth]# trust 28:CD:C1:FF:B7:8D
[bluetooth]# exit
```

(you may need to be in the bluetooth group: `sudo usermod -aG bluetooth $USER`)

- macOS: System Settings -> Bluetooth -> click "Connect" next to PicoMite-XXXX. Accept the "Terminal would like to use Bluetooth" prompt the first time ble_term runs.

Run

```
python ble_term.py PicoMite-77BA
```

(or just python ble_term.py for the default Pico-RP2350 name.)

You'll see a banner with the available commands, then ble_term scans for the device, connects, and you get the MMBasic prompt directly in the same window.

In-terminal commands

ble_term uses Ctrl-] (Telnet-style escape) as its command prefix -- press it once, then a letter:

Key	Command
q, x	Quit ble_term
s	XMODEM send a host file to the Pico
r	XMODEM receive a file from the Pico to the host
?, h	Show command help

Any other key after Ctrl-] cancels the command and returns to normal mode. Ctrl-C alone is forwarded to MMBasic as the BASIC break key.

XMODEM file transfer

ble_term implements XMODEM-CRC (128-byte packets) directly over the BLE link -- no external file-transfer tool required. The workflow mirrors the Tera Term style but driven from the one window:

Send a file to the Pico:

1. At the MMBasic prompt, type:

```
XMODEM R progname
```

The Pico starts emitting NAKs (you'll see them in the terminal).

2. Press Ctrl-] then s. ble_term prompts:

```
[xmodem-send] host file to send:
```

3. Type the host path (Backspace and ESC work; ESC cancels), press Enter.
4. Progress is shown as [xmodem] sent N/N bytes, then [xmodem] send complete. Keyboard input is ignored during the transfer.

Receive a file from the Pico:

1. At the MMBasic prompt, type:

```
XMODEM S progname
```

The Pico starts sending packets.

2. Press Ctrl-] then r. ble_term prompts for the host file to save into.
3. Type the host path, press Enter. Progress as `[xmodem] received N bytes, then [xmodem] receive complete`.

The transfer uses the same chunked-write/retry logic as the bridge, so it works through transient BLE hiccups.

Auto-reconnect

If the Pico reboots (e.g. OPTION CPU_SPEED change or watchdog), ble_term detects the BLE disconnect, scans, and reconnects automatically. Keystrokes typed during the reconnect window are buffered and delivered as soon as the link is back. Each session prints a numbered status line on stderr so you can see what's happening:

```
[ble_term] #1 connected, notify subscribed
...
[ble_term] #1 disconnected (received 482 notifies this session)
[ble_term] session #2 scanning for 'PicoMite-77BA'...
[ble_term] #2 connecting to 28:CD:C1:FF:B7:8D
[ble_term] #2 connected, notify subscribed
```

Key translation

ble_term translates a few host-terminal keystrokes to the forms MMBasic's MMInkey parser recognises:

Host key	Sent to Pico	Effect
Backspace (host sends 0x7F)	0x08 (BS)	Deletes previous char
Home (\033[H or \033OH)	\033[1~	Cursor to start of line
End (\033[F or \033OF)	\033[4~	Cursor to end of line

Arrow keys, Delete, Insert, PgUp/PgDn, F1-F12 already match what MMBasic expects, so they pass through untouched. If you find a key that doesn't behave, the fix is to add a mapping in translate_keys() in ble_term.py.

Alternative -- ble_bridge.py with an external terminal

If you prefer using your existing terminal (Tera Term on Windows, PuTTY on Linux, telnet on macOS) instead of ble_term, ble_bridge.py forwards BLE NUS over TCP and you connect a Telnet client to 127.0.0.1:5555. The bridge speaks full Telnet NVT (IAC doubling, CR-NUL stuffing) so XMODEM and YMODEM work the same as in WEB build.

The detailed per-OS setup follows. **If you're new here, use ble_term above -- it's simpler and covers all three OSes from one script.**

Windows (Tera Term)

Windows doesn't expose a BLE serial port as a COM port, so we run a small Python bridge that converts BLE NUS into a TCP socket, then point Tera Term at that socket.

One-off setup

1. Pair the device once via Windows Settings:
- Settings -> Bluetooth & devices -> Add device -> Bluetooth
- Pick PicoMite-XXXX from the list, wait for "Paired".
- Windows will show it as "Not connected" -- that is expected; the Settings UI only handles pairing, not the GATT data stream.

2. Install Python and bleak:

```
pip install bleak
```

3. Configure Tera Term (one-off; Setup -> Save setup... persists these):
- Setup -> Terminal -> Local echo: OFF
- Setup -> Terminal -> Local Edit Mode: OFF
- Setup -> Terminal -> Terminal ID: VT100 (or VT-AT386)

Every session

1. Start the bridge:

```
python ble_bridge.py PicoMite-XXXX
```

The bridge scans, connects, then listens on TCP port 5555: ` [TCP] listening on telnet://127.0.0.1:5555 [BLE] target device: 'PicoMite-XXXX' [BLE] scanning for 'PicoMite-XXXX'... [BLE] found 28:CD:C1:..., connecting... [BLE] connected, subscribing to TX notifications `

2. In Tera Term: File -> New connection -> TCP/IP
- Host: 127.0.0.1
- Service: Telnet
- TCP port#: 5555

3. Hit Enter, you get the MMBasic prompt.

Bridge options

```
python ble_bridge.py                       # default device name "Pico-RP2350"
python ble_bridge.py PicoMite-77BA         # specific device
python ble_bridge.py PicoMite-77BA --port 6000
```

Auto-reconnect

If you reboot the Pico (OPTION CPU_SPEED, error trap, manual reset) the bridge automatically reconnects within ~10 seconds without dropping the Tera Term TCP session. You'll see this in the bridge console:

```
[BLE] disconnected
[BLE] scanning for 'PicoMite-77BA'...
[BLE] found ..., connecting...
[BLE] connected, subscribing to TX notifications
```

Tera Term stays put; data flow pauses then resumes.

Linux (PuTTY)

Same architecture as Windows: pair once, then run ble_bridge.py and point a terminal at TCP 127.0.0.1:5555.

One-off setup

1. Install bleak and PuTTY:

```
pip install bleak
sudo apt install putty            # or: dnf install putty
```

2. Pair the device once via bluetoothctl:

```
$ bluetoothctl
[bluetooth]# scan on
[NEW] Device 28:CD:C1:FF:B7:8D PicoMite-77BA
[bluetooth]# pair 28:CD:C1:FF:B7:8D
[agent] Confirm passkey ... (yes/no): yes
[bluetooth]# trust 28:CD:C1:FF:B7:8D
[bluetooth]# scan off
[bluetooth]# exit
```

The bond is stored in /var/lib/bluetooth/<adapter>/<peer-mac>/.

3. User permissions -- your user must be allowed to use BLE:

```
sudo usermod -aG bluetooth $USER
```

Log out and back in. (Some distros handle this via polkit; if you see DBusError: Not authorized from bleak, this is what to fix.)

Every session

1. Start the bridge:

```
python ble_bridge.py PicoMite-77BA
```

2. Open PuTTY:
- Session -> Connection type: Telnet
- Host Name: 127.0.0.1
- Port: 5555
- Open

Telnet mode is required (Raw mode does NOT work) because the bridge speaks standard NVT Telnet: it negotiates char-at-a-time + server echo via IAC options, doubles 0xFF data bytes as IAC IAC, and stuffs 0x0D (CR) data bytes as CR NUL. PuTTY's Telnet code handles all of that correctly; Raw mode would expose the NUL stuffing and IAC bytes as garbage characters and break XMODEM.

3. Set vt100 emulation: in the PuTTY config tree, Terminal -> Keyboard
leave defaults; **Window -> Translation -> Remote character set: UTF-8; Window -> Colours -> ANSI Colour:** OK.

4. Save the session (give it a name, click "Save") so you don't have to
re-enter the settings.

---

macOS (telnet)

Same architecture as Windows and Linux: pair once, then run ble_bridge.py and connect a Telnet terminal to TCP 127.0.0.1:5555.

One-off setup

1. Install bleak and a Telnet client. macOS removed the built-in
telnet binary in 10.15+, so install one via Homebrew: ` pip3 install bleak brew install telnet # or: brew install inetutils ` (Both packages provide a telnet command.)

2. Pair the device once via the GUI:
- Open System Settings -> Bluetooth.
- Wait for PicoMite-XXXX to appear in the list of nearby devices.
- Click Connect next to it. macOS auto-pairs BLE devices using
Just Works -- no prompt for a PIN.
- The device should then show as Connected briefly and drop to
Not Connected once macOS finishes pairing. That's expected; the GUI doesn't keep an active GATT session.

3. bleak permissions. Modern macOS prompts for Bluetooth access
the first time bleak is used. When you run ble_bridge.py, look for a "Terminal would like to use Bluetooth" alert and accept it. You can also pre-authorise under **System Settings -> Privacy & Security -> Bluetooth -> Terminal (or your Python interpreter)**.

Every session

1. Start the bridge:
```
python3 ble_bridge.py PicoMite-77BA
```

2. In another Terminal window:
```
telnet 127.0.0.1 5555
```

telnet on macOS handles NVT options out of the box, so char-at-a-time mode and CR-NUL stuffing both Just Work -- same reasoning as Windows Tera Term and Linux PuTTY in Telnet mode.

If you'd rather use a graphical terminal, iTerm2 doesn't have a built-in Telnet client, but you can run telnet 127.0.0.1 5555 inside it. CoolTerm and SerialTools are serial-only and won't work without further glue.

3. To exit telnet, type Ctrl-] then quit.

Do not use nc (netcat) -- it's raw TCP and doesn't understand Telnet NVT negotiation, so the keyboard ends up line-buffered and XMODEM transfers break on CR-NUL stuffing.

Pairing model and key storage

PicoMiteBT uses BLE Just Works pairing -- no PIN, no MITM protection. Pairing is automatic; the user doesn't have to type or confirm anything beyond the OS-level "Pair?" dialog.

After pairing, the long-term key (LTK) is stored on the Pico inside the PicoMite Option struct (Option.bt_tlv[] -- a 2 KB area split into two 1 KB banks). It survives reboot and OPTION RESET -- only a full factory reset / MagicKey change wipes it.

Up to 4 hosts can be paired at the same time. The Pico keeps a key for each. So you can pair the same Pico with your desk PC, laptop, and phone, and reconnect from whichever one is convenient without re-pairing.

When the table is full and a 5th host pairs, btstack evicts the oldest entry; that host then has to re-pair next time.

Limitations

These are constraints of the design, not bugs to be reported.

One active connection at a time

BLE peripherals advertise to many scanners but accept only **one active connection**. So while you can be paired with PC + phone + laptop at once, only one of those can have an open session at any moment. The others can't connect until the current one disconnects.

No virtual COM port on Windows

BLE NUS is a GATT profile, not Bluetooth Classic SPP. Windows does not auto-create a COM port for it, and no terminal app talks BLE NUS natively. That's why the Python bridge exists -- it presents a TCP socket that Tera Term/PuTTY can attach to.

If you really want a COMxx device for compatibility with other software, you'd need a virtual-COM driver like com0com and a small custom helper. The bridge could be extended to drive com0com instead of (or in addition to) TCP. Not included.

iOS pairing nuance

iOS will not allow BLE pairing without an active GATT operation following soon after. If you "pair" without then connecting, iOS may silently un-pair. Always connect from an app right after pairing.

Reconnect delay

Rebooting the Pico (OPTION CPU_SPEED, watchdog reset, etc.) means ~10 seconds before the bridge reconnects.

Breakdown of where the time goes:

- Pico firmware boot + cyw43 BT firmware download: ~3-5 s
- HCI bring-up + first advertisement: ~1 s
- Host's BLE scan latency: ~2-5 s
- Bridge retry backoff: 1 s

Nothing to do about the Pico-side cost; the bridge could be extended to cache the BD_ADDR and reconnect by address (skipping the scan) for ~3-5 s savings.

Bond mismatch after firmware rebuild

A firmware build that bumps MagicKey triggers ResetOptions() on first boot, which zeroes Option.bt_tlv. The Pico forgets its bonds but the host doesn't. Result on reconnect:

- Windows: characteristic reads return "Unreachable" until you Settings -> Remove device -> re-pair.
- Linux: "Authentication failed" -- fix is bluetoothctl remove <MAC> then re-pair.
- Phone: Forget the device in Bluetooth settings, re-pair.

This only happens after MagicKey changes (i.e., development builds that change the Option layout). Stable releases don't disturb existing bonds.

No persistence on PSRAM-only boards

The bond storage lives in the PicoMite Option flash sector. Boards where PicoMite runs from PSRAM still write Options to flash, so this is not a real limitation -- mentioned only because it's a common question.

Throughput

RFCOMM-class throughput is not the goal -- this is a console link. Typical sustained throughput is ~5-20 KB/s depending on the host's BLE stack and connection parameters. Fast enough for typing, listing programs, and both XMODEM and YMODEM transfers (YMODEM in particular is a strict 8-bit-clean test of the whole pipeline and runs through cleanly at 200 MHz CPU). Not suited for moving large binary files end-to-end.

CPU speed bounds

The cyw43 wireless driver and btstack run in alarm-IRQ context. At low CPU speeds, the per-event processing margin becomes too tight for sustained bidirectional traffic and the link destabilises. To prevent this the build enforces a CPU-speed range:

- MIN_CPU = 200 MHz -- the safe floor for BLE on RP2350. Below this the BLE stack can't reliably keep up with AutoSave paste-ins or XMODEM/YMODEM transfers.
- MAX_CPU = 396 MHz -- comfortable overclock headroom for RP2350-A on Pico 2 W. Lower than the standard PicoMite ceiling so the cyw43 SPI link stays well within its 50 MHz rating even when the PIO divider scaling rounds up.

ResetOptions (triggered on first boot after flashing, or after any MagicKey change) sets Option.CPU_Speed to 200 MHz. You can adjust within bounds via OPTION CPU but the firmware rejects values outside [200000, 396000].

When pasting very large programs over BT, prefer AUTOSAVE N (no-echo). The echo path roughly doubles BLE traffic; suppressing it halves the Pico's per-byte workload and tightens timing margins.

Range

Practical ranges to expect, governed by the CYW43439 radio (Class 2 BLE, ~+4 dBm TX, ~-93 dBm RX) and the small

PCB antenna on Pico 2 W:

Scenario	Realistic range
Open air, line of sight	20-30 m
Same room with normal furniture	10-15 m
Through one interior wall (drywall)	5-10 m
Through two walls or one concrete wall	2-5 m, often unreliable
Between floors	3-7 m, marginal

Things that improve range:

- Class 1 USB Bluetooth dongle on the host (up to +20 dBm) -- adds several meters, especially through obstructions, because the better receiver on the host side carries the weak Pico transmissions.
- Line of sight. One drywall costs ~10 dB, two walls ~20 dB -- that's most of the link budget. A hallway goes much further than the same distance through walls.
- Antenna orientation. The PCB antenna has lobes; rotating the board 90° can rescue a borderline link. Standing up beats flat on a desk.

Things that don't help much in practice:

- BLE 2M PHY (higher throughput, lower range -- we use 1M which is better balanced for this use).
- BLE Coded PHY (long-range Bluetooth 5) -- neither btstack here nor Windows BLE negotiate it readily for NUS; not configured.
- External antenna mod -- the antenna pad/disable resistor exists on Pico 2 W (like Pico W), but the mod is fiddly and only adds a few dB.

In short: plenty of range for working at a desk, debugging from across the bench, or running BASIC from a phone in the same room -- but don't expect to control devices reliably from another floor of the building.

BLE only -- no Bluetooth Classic

The build links pico_btstack_ble, not pico_btstack_classic. There is no SPP, no audio (A2DP), no HID. Pure BLE GATT for the NUS profile.

Console output buffering during pre-pairing

Anything PicoMite prints before a host has subscribed to TX notifications is buffered in a 1 KB ring on the device. If the device prints more than 1 KB before you connect, the oldest bytes are dropped to make room for newer ones -- so the most recent output (the prompt) is what reaches you when you finally connect.

Troubleshooting

"Could not start notify on 000A: Unreachable" (bleak / Windows)

The host has a cached LTK that doesn't match the Pico's. Remove the device from Windows Settings -> Bluetooth & devices, then pair again.

Bridge keeps cycling connect/disconnect

Same root cause as above. Each cycle is the host attempting reconnect with a stale LTK. Re-pair fixes it.

"Authentication failed" (Linux)

bluetoothctl remove <PicoMAC> then re-pair (scan on, pair, trust).

"DBusError: Not authorized" (Linux)

User isn't in the bluetooth group. sudo usermod -aG bluetooth $USER then log out / back in.

Tera Term shows weird block characters / nothing

- Setup -> Terminal -> Local Edit Mode must be OFF.
- Service must be Telnet (the bridge sends IAC negotiation that Tera Term needs to switch to char-at-a-time mode).
- Local echo must be OFF (PicoMite echoes; turning local echo on doubles every keystroke).

Characters only echo when I press Enter

Tera Term Local Edit Mode is on. Turn it off; save setup.

Connecting from a fresh laptop, BLE LE Explorer shows the device

but characteristics are "Unreachable" The device needs to be paired through OS settings first. BLE Explorer itself cannot drive pairing on Windows; it only browses. Pair via Settings -> Add device, then re-open Explorer.

Bridge prints "[BLE] not found, will retry"

Pico may have rebooted; wait ~10 s. If it persists, check that the device is still advertising -- connect from a phone to verify, or re-flash the firmware. Could also be Windows BLE permissions / adapter state: in Settings -> Bluetooth, toggle Bluetooth off then on to reset the host stack.

"no such file or directory" from nmake in CMake build

Not BLE-related, but a common gotcha: Visual Studio's MSVC tools aren't on PATH. Launch VS Code from the **Developer Command Prompt for VS**, or CMake: Select a Kit and choose the Visual Studio kit.

Files of interest in this build

File	Purpose
BTConsole.c / BTConsole.h	BLE NUS server, TX/RX rings, GATT/SM hooks
nus_gatt.gatt	GATT database (NUS UUIDs, characteristics, flags); compiled to nus_gatt.h by btstack's compile_gatt.py
btstack_config.h	btstack feature flags; controls bonding, encryption, max peer count
ble_term.py	Cross-platform self-contained BLE NUS terminal with built-in XMODEM (recommended desktop path)
ble_bridge.py	Alternative cross-platform BLE NUS -> TCP forwarder for users who prefer an external Telnet client
FileIO.h	Defines Option.bt_tlv[2048] for persistent bond storage
configuration.h	PICOMITEBT block: flash offsets, magic key, heap sizes

PicoMiteBT -- BLE Console for Pico 2 W

CMakeLists.txt

PICOBTRP2350 variant definition; links pico_btstack_ble, pico_btstack_cyw43, pico_cyw43_arch_none