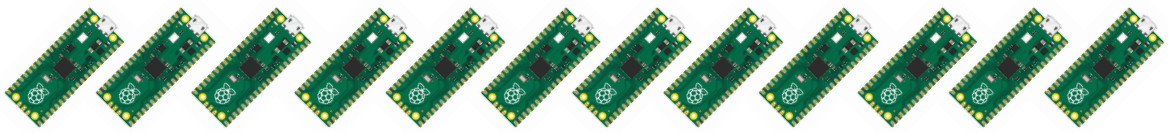
Arduino IDE's **Compile–Upload–Run** for Raspberry Pi Pico

Dong Heon Han

1/1/2026



1 Overview

This document describes the complete execution pipeline of the **Arduino development environment**, including compilation, upload, and runtime execution. The behavior is explained for both macOS and Windows hosts, with a particular focus on USB-native microcontrollers such as the RP2040 (Raspberry Pi Pico). Importantly, **no user code is executed during compilation or upload**. User code begins execution only after a successful upload and device reset.

2 Phase I: Compilation and Linking (Host Side)

During compilation, the Arduino build system performs the following steps:

1. Convert all `.ino` files into valid C++ source files and insert function prototypes automatically.
2. Compile all source files and libraries into object files. Construct `.cpp, .h`
3. Link all object files into an executable (`.elf`).
4. Convert the executable into an uploadable image (RP Pico uses `.uf2`).

Key fact. During this phase, neither `setup()` nor `loop()` is executed. Compilation occurs entirely on the host computer.

3 Phase II: Upload Procedure (macOS vs Windows)

After compilation, the host attempts to upload the generated binary to the target device. The mechanism depends on the board family. **3.1 is written to compare the method with 3.2 (our method)**.

3.1 Serial bootloader based boards (Arduino UNO, Mega)

For boards using a serial bootloader, the host:

1. Opens the serial port (macOS: `/dev/cu.*`, Windows: `COM*`).
2. Toggles **reset** using DTR/RTS or a 1200-baud "touch" sequence. ("touch": open/close procedure), bootloader phase 1)
3. Transfers the binary using a flashing utility (e.g., `avrdude`, bootloader phase 2).

```
** bootloader phase 1
Host side:
- open serial port at 1200 baud
- immediately close the port (touch)
- trigger auto-reset via DTR/RTS or CDC event

Target side (firmware / ROM):
- detect 1200-baud open/close or DTR/RTS toggle
- perform software or hardware reset
- jump to bootloader
- start listening for programming commands

-------------------------------------------------
** bootloader phase 2 (flash)
Host side:
- avrdude opens port
- handshake with bootloader (packet send 0x30, 0x20 :example)
- send erase / write / verify packets

Target side (bootloader firmware):
- receive packets
- parse protocol
- erase flash pages
- program flash
- send ACK / status
```

3.2 RP2040 / Pico UF2-based upload

For RP2040-based boards, the upload procedure is typically:

1. The host triggers a reset into the bootloader (often via a 1200-baud "touch" on the USB CDC port, if supported by the core/toolchain).
2. The board disconnects from USB and reconnects as a **USB Mass Storage (UF2) bootloader**.
3. A new removable drive appears:
   - macOS: mounted under `/Volumes/RPI-RP2`
   - Windows: assigned a new drive letter (e.g., `E:`)
4. The host copies the generated `.uf2` file to the `RPI-RP2` drive.
5. The UF2 bootloader **parses the UF2 blocks and performs flash programming** (erase/write/verify), then resets the board to start user firmware. **(flash)** [1]

[1] Flash is a non-volatile program memory that retains the firmware after power-off. After upload, the compiled binary (`.uf2 / .bin`) is written into flash memory, and after reset the CPU fetches and executes instructions directly from flash.

```
** UF2 bootloader entry (phase 1: enter boot mode; not flash)
Host side:
- open USB CDC port (if present)
- request bootloader mode (often via 1200-baud open/close "touch", or vendor/core
    reset request)
- device disconnects from USB

Target side (RP2040 ROM / second-stage bootloader):
- detect boot request (CDC touch or explicit reset path)
- reset MCU into BOOTSEL/UF2 mode
- enumerate as USB Mass Storage device (RPI-RP2)
- expose a virtual FAT filesystem to the host

-------------------------------------------------
** UF2 flashing (phase 2: flash programming)
Host side:
- OS copies <firmware>.uf2 onto the RPI-RP2 drive
- host "file write" is translated into 512-byte UF2 blocks

Target side (UF2 bootloader):
- receive UF2 blocks via USB Mass Storage writes
- validate UF2 headers and target addresses
- erase flash sectors as needed
- program flash with the UF2 payload data
- (optional) verify written contents / checks
- finalize programming, then reboot

-------------------------------------------------
** reboot into user code
Target side:
- exit UF2 mode
- reset
- execute user firmware from flash (XIP)
```

If the bootloader drive does not appear, the upload fails with errors such as:

```
Scanning for RP2040 devices
No drive to deploy.
```

4 Phase III: Runtime Execution Model

After a successful upload, the microcontroller resets and begins executing code from flash. The Arduino core provides a C/C++ entry point `main()` that invokes the user-defined `setup()` and `loop()` functions.

4.1 Explicit Arduino `main()` Implementation (important!)

A simplified and platform-independent representation of the Arduino runtime is shown below.

```c
#include <Arduino.h>

/* Forward declarations of user functions */
void setup(void);
void loop(void);

/* Arduino runtime entry point */
```

```
int main(void)
{
    /* Core and hardware initialization */
    init(); // clocks, timers, peripherals, USB

#if defined(USBCON) || defined(ARDUINO_ARCH_RP2040)
    /* Attach USB device on USB-native platforms */
    USBDevice.attach();
#endif

    /* User initialization: executed exactly once */
    setup();

    /* Infinite execution loop */
    for (;;) {
        loop(); // user task
        yield(); // background servicing (USB, WiFi, etc.)
    }

    /* This point is never reached */
    return 0;
}
```

Execution semantics.

- `main()` is entered only after a successful upload and reset.
- `setup()` is executed exactly once.
- `loop()` is executed repeatedly in an infinite loop.
- No user code executes during compilation or upload.

5 USB Starvation and Upload Failures

On USB-native boards (including RP2040), the USB stack requires periodic servicing. If the user code implements a tight busy loop without any `delay()` or `yield()`, USB background tasks may be starved.

In this case, the automatic reset into the bootloader used for subsequent uploads may fail, preventing the UF2 drive from appearing and causing upload errors.

5.1 Recommended Safe Polling Pattern

A minimal safe pattern for serial polling is:

```
void loop() {
    if (Serial.available() < NEEDED_BYTES) {
        delay(1); // or yield();
        return;
    }
    /* process incoming data */
    delay(1);
}
```

6 Manual Recovery

If automatic upload fails, force the bootloader manually:

1. Disconnect the USB cable.
2. Hold the BOOTSEL button on the board.
3. Reconnect the USB cable while holding the button.
4. Upload again or copy the `.uf2` file manually.

7 Messages from Arduino IDE's Output

This section illustrates typical messages printed by the Arduino IDE during the upload process. The outputs are divided into **normal (successful) uploads** and **error cases**, with explanations of the meaning of each line.

7.1 Successful upload (RP2040 / UF2-based boards)

A typical successful upload log for an RP2040-based board is shown below:

```
Sketch uses 412356 bytes (15%) of program storage space.
Global variables use 51234 bytes (19%) of dynamic memory.

Scanning for RP2040 devices
Found drive at /Volumes/RPI-RP2
Uploading firmware.uf2
Done.
```

Explanation.

- `Sketch uses ...` Indicates the size of the compiled program stored in flash memory.
- `Global variables use ...` Reports the amount of SRAM used for global/static variables.
- `Scanning for RP2040 devices` The host is searching for an RP2040 board that has entered UF2 bootloader mode.
- `Found drive at /Volumes/RPI-RP2` The board has successfully re-enumerated as a USB Mass Storage bootloader.
- `Uploading firmware.uf2` The host copies the UF2 file to the bootloader drive. This triggers flash programming.
- `Done.` Flash programming has completed successfully and the board has rebooted into user code.

7.2 Successful upload (Serial bootloader based boards)

A typical successful upload log for Arduino UNO/Mega boards is shown below:

```
Sketch uses 12345 bytes (38%) of program storage space.
Global variables use 456 bytes (22%) of dynamic memory.

avrdude: Version 6.3
avrdude: Opening serial port /dev/cu.usbmodem14101
avrdude: Sending reset command
avrdude: stk500_getsync() OK
avrdude: Writing | ################################################## | 100% 1.23s
avrdude: Verifying ...
avrdude: 12345 bytes of flash written
avrdude done. Thank you.
```

Explanation.

- `Opening serial port ...` The host opens the serial port connected to the target board.
- `Sending reset command` The host toggles DTR/RTS to reset the board and enter the bootloader.
- `stk500_getsync() OK` The host has successfully synchronized with the serial bootloader.
- `Writing ... 100%` The bootloader is erasing and programming the flash memory.
- `Verifying ...` The programmed flash contents are being verified.
- `avrdude done.` Flash programming has completed successfully and the board will reboot into user code.

7.3 Common error case: Bootloader not detected (RP2040)

If the board does not enter UF2 bootloader mode, the following error may occur:

```
Scanning for RP2040 devices
No drive to deploy.
Error: Could not find RP2040 device
```

Explanation.

- `Scanning for RP2040 devices` The host is attempting to detect a board in UF2 bootloader mode.
- `No drive to deploy` No USB Mass Storage device corresponding to the RP2040 bootloader was found.
- `Error: Could not find RP2040 device` The upload fails because the board did not enter bootloader mode.

7.4 Common error case: Serial synchronization failure

For serial bootloader boards, a typical synchronization error is:

```
avrdude: stk500_getsync(): not in sync: resp=0x00
avrdude: stk500_recv(): programmer is not responding
```

Explanation.

- `not in sync` The host failed to establish communication with the bootloader.
- `programmer is not responding` The board did not enter bootloader mode or the wrong serial port/baud rate was selected.

7.5 Recovery hint

In most error cases, the upload can be recovered by:

- Manually resetting the board into bootloader mode (e.g., holding the BOOTSEL button on RP2040).
- Verifying the correct serial port and board type are selected in the Arduino IDE.
- Closing any program that is currently using the serial port.

References

[1] Adafruit. Uf2 bootloader details. `https://learn.adafruit.com/uf2-bootloader-details`, 2024.

[2] Arduino. Arduino bootloader. `https://docs.arduino.cc/learn/programming/arduino-bootloader`, 2024.

[3] Arduino. The arduino software (ide) and build process. `https://docs.arduino.cc/learn/starting-guide/the-arduino-software-ide`, 2024.

[4] Arduino. Uploading using the 1200-baud reset mechanism. `https://github.com/arduino/ArduinoCore-samd/wiki/Uploading`, 2024.

[5] Arduino Core AVR. main.cpp – arduino runtime entry point. `https://github.com/arduino/ArduinoCore-avr/blob/master/cores/arduino/main.cpp`, 2024.

[6] Microchip Technology. Avr stk500 communication protocol. Technical report, Microchip, 2016.

[7] Microsoft. Uf2 bootloader format specification. `https://github.com/microsoft/uf2`, 2024.

[8] Raspberry Pi Ltd. Getting started with raspberry pi pico. `https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html`, 2024.

[9] Raspberry Pi Ltd. Raspberry pi pico datasheet. `https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf`, 2024.

[10] Raspberry Pi Ltd. Rp2040 datasheet. `https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf`, 2024.

[11] Savannah Project. avrdude – avr downloader/uploader. `https://www.nongnu.org/avrdude/`, 2024.