Arduino EasyScript

English-like code → Arduino C++ (Arduino core only).

What you get
• Minimal web IDE with Monaco (autocomplete, hover help, formatter, outline).
• A small language that compiles into a normal .ino sketch.
• Friendly errors/warnings (missing end, unknown commands, pin range, etc.).

Version: 2.0.0 **Date:** 2026-02-12

This PDF documents the rebuilt project in EngToArd_Rebuild/. Everything maps to Arduino core APIs (no extra Arduino libraries required).

Quickstart

Run a local server (Monaco uses web workers, so opening the HTML file directly is unreliable).

Run locally with Python

```
cd EngToArd_Rebuild
python -m http.server 8000
# open http://localhost:8000
```

Write EasyScript on the left → generated Arduino C++ appears on the right. Fix errors (red) and optionally warnings (yellow), then download sketch.ino.

> **Workflow:** EasyScript → transpile → download .ino → open in Arduino IDE → upload to board.

Language overview

AES is line-based and uses blocks. Blocks end with end (no braces). Whitespace and capitalization are flexible.

Comments

Use # or // for single-line comments.

Comments

```
# comment
// also a comment
```

Core blocks

Blocks

```
setup:
  # runs once
end

loop:
  # runs forever
end

function blink:
  # reusable code
end

if x is 0 do:
  print "pressed"
otherwise do:
  print "released"
end
```

Named pins

Pins are defined at the top using pin NAME is VALUE. Then you can use NAME everywhere.

Pin declarations

```
pin led is 13
pin pot is A0
pin button is 2
```

Using named pins

```
make led output
read analog pot into raw
turn led on
```

Variables + expressions

Variables are created automatically. Expressions use Arduino/C++ operators.

Variables

```
set counter to 0
change counter by 1
set x to (counter * 2) + 10
```

Timers without delay()

every and after compile to millis()-based checks.

Non-blocking timers

```
every 250 ms do:
  toggle led
end

after 2 seconds do:
  print "go!"
end
```

Command reference

All commands below are implemented in the rebuild and compile to Arduino core functions.

pin NAME is VALUE

Defines a named pin (VALUE: 0..13 or A0..A5 on UNO preset). Top-level only.

Pins compile to const integers in generated C++.

```
pin led is 13
pin pot is A0
```

make PIN output | input | input pullup

Sets pinMode for a pin.

```
make led output
make button input pullup
```

turn PIN on | off

Writes HIGH/LOW to a digital pin.

```
turn led on
turn led off
```

toggle PIN

Flips the digital state (digitalRead + digitalWrite).

```
toggle led
```

read PIN into VAR

Reads a digital pin (0/1) into a variable.

```
read button into pressed
```

read analog PIN into VAR

Reads an analog pin (A0..A5) into a variable.

Using a digital pin here is a compiler error.

```
read analog pot into raw
```

read time into VAR

Stores millis() into a variable.

```
read time into now
```

read micros into VAR

Stores micros() into a variable.

```
read micros into t
```

set PIN to pwm VALUE

Writes PWM value 0..255 (analogWrite).

```
set led to pwm 128
set led to pwm brightness
```

wait N ms | seconds | us

Blocks using delay() (ms/seconds) or delayMicroseconds() (us).

For non-blocking timing, prefer every/after.

```
wait 250 ms
wait 1 second
wait 50 us
```

every N ms | seconds do:

Runs a block repeatedly without delay().

```
every 500 ms do:
  toggle led
end
```

after N ms | seconds do:

Runs a block once after a delay without blocking the loop.

```
after 2 seconds do:
  print "go!"
end
```

seed random with EXPR / random seed EXPR

Seeds random() generator (randomSeed).

```
seed random with 123
pick random from 0..10 into r
```

start serial at BAUD

Starts Serial (Serial.begin).

```
start serial at 9600
```

print …

Prints to Serial Monitor. Supports strings, expressions, and: print "label" and x

```
print "hello"
print x
print "x=" and x
```

set VAR to EXPR

Assigns a value to a variable.

```
set x to 0
set x to (x + 1) * 2
```

change VAR by EXPR

Adds a delta to a variable.

```
change x by 1
change x by -5
```

map SRC from a..b to c..d into DEST

Uses Arduino map() to scale a value.

```
map raw from 0..1023 to 0..255 into brightness
```

limit VAR to lo..hi

Clamps a value into a range (like constrain).

```
limit brightness to 0..255
```

pick random from a..b into VAR

Picks a random integer (random()).

```
pick random from 0..100 into r
```

play tone FREQ on PIN / stop tone on PIN

Starts/stops a buzzer tone (tone/noTone).

```
play tone 440 on buzzer
wait 200 ms
stop tone on buzzer
```

call NAME

Calls a function defined with function NAME: … end

```
call blink
```

Editor features (Monaco)

The rebuild keeps Monaco but makes it feel language-aware:

Context-aware autocomplete (pins, modes, functions, variables).
Auto-insert end when you press Enter after setup:/loop:/if/every/etc.
Hover docs for core keywords (make, every, wait, seed...).
Formatter (Format button or Shift+Alt+F).
Outline + Go to Definition for functions and pins.
Problems panel (click to jump).

Examples

These are included in the app’s example dropdown.

Blink

```
pin led is 13

setup:
  make led output
end

loop:
  turn led on
  wait 500 ms
  turn led off
  wait 500 ms
end
```

Non-blocking blink (every)

```
pin led is 13

setup:
  make led output
end

loop:
  every 250 ms do:
    toggle led
  end
end
```

Extending AES

Add new commands in src/aes/service.js inside the COMMANDS array. Then update autocomplete/hover in src/monacoAes.js and (optionally) add examples.

One place for command handlers

```
const COMMANDS = [
  function cmd_make(code, lineNo){ ... },
  // ...
  // add yours here
];
```

Recommended style rules:

• Keep syntax predictable and one-command-per-line.
• Prefer warnings for board-specific assumptions (like pin ranges).
• When rejecting something, explain how to fix it.