

# Introduction

## Introduction

Céu-Arduino supports the development of Arduino applications in the programming language Céu.

## Modes of Operation

### Modes of Operation

A mode of operation specifies how Céu-Arduino captures events from the environment (e.g., pin changes) and redirects them to the Céu application.

Céu-Arduino supports the *polling* and *interrupt-based* modes of operation.

The polling mode is the default mode of operation.

The modes of operation are implemented in C and are part of Céu-Arduino. Each mode is described in pseudo-code as follows.

### Polling

The *polling mode* of Céu-Arduino continually checks for changes in the environment in an infinite loop:

```
void setup () {
    ceu_start();
    while (<program-is-running>) {
        ceu_input(CEU_INPUT__NONE, NULL, <time>);    /* input async and timer */
        if (<pin--changed>) {
            ceu_input(CEU_INPUT__PIN_*, <...>);      /* input pins */
        }
        <...>    // for each pin
    }
    ceu_stop();
    while (1);                                     /* freezes arduino */
}

void loop () { /* never reached */ }
```

The inputs are polled on each loop iteration and changes are notified to the Céu application through `ceu_input` calls.

The polling mode uses *100%* of the CPU time.

## Input Events

Currently, the polling mode supports the following input events:

- Timers
- Asynchronous blocks
- Digital pins

## Compilation

Since polling is the default mode of operation, compilation only needs to provide the Céu application:

```
$ make CEU_SRC=<path-to-ceu-application>
```

## Interrupts

In the *interrupt-based mode* of Céu-Arduino, all input is done in Céu itself through `async/isr` blocks. Emitting an input event from an `async/isr` only sets a flag which is then checked in the Arduino loop:

```
void setup () {
    ceu_start();
    while (<program-is-running>) {
        ceu_input(CEU_INPUT__NONE, NULL, CEU_WCLOCK_INACTIVE);
        if (<any-isr-evt-occurred>) {                // interrupts off
            ceu_input(<isr-evt-occurring>, <...>);    // interrupts on
        }
#ifdef CEU_FEATURES_ISR_SLEEP
        else if (!<program-has-pending-async>) {
            <enter-sleep-mode>
        }
#endif
    }
    ceu_stop();
    while (1);                                     /* freezes arduino */
}

void loop () { /* never reached */ }
```

To comply with the synchronous semantics of Céu, all `ceu_input` calls are serialized in the loop.

If the macro `CEU_FEATURES_ISR_SLEEP` is defined, the Arduino enters in the `SLEEP_MODE_IDLE` sleep mode after each reaction.

Interrupts are disabled only while checking for occurring inputs. Hence, `async/isr` blocks and synchronous code may be concurrent and require `atomic`

blocks.

An `async/isr` in Céu-Arduino requires two arguments:

- the interrupt number (i.e., the index in the interrupt vector)
- the interrupt trigger mode (i.e., when the interrupt should be triggered)

The interrupt trigger mode is only used for digital pin interrupts:

<https://www.arduino.cc/en/Reference/AttachInterrupt>

The example that follows executes the code marked as `<...>` whenever the value of *pin 2* changes:

```
spawn async/isr [_digitalPinToInterrupt(2),_CHANGE] do
  <...>
end
```

## Input Events

Drivers:

- `pin-02`: TODO
- `timer`: TODO
- `usart`: TODO

## Compilation

Applications that use interrupts have to be compiled with `CEU_ISR=true`:

```
$ make CEU_ISR=true CEU_SRC=<path-to-ceu-application>
```

# Digital Pins

## Digital Pins

Céu-Arduino supports `emit` and `await` statements for digital pins in output and input modes, respectively.

## Input Pins

A program can `await` a change in a digital pin configured as input and acquire its current value:

```
input int PIN_02;
var int v = await PIN_02;
```

In the interrupt mode, the pin requires a driver to generate the input:

```
##include "arduino/isr/pin-02.ceu"  
input int PIN_02;  
var int v = await PIN_02;
```

## Output Pins

A program can **emit** a change to a digital pin configured as output.

### Digital Output

For digital output, the pin number requires the prefix `PIN_`:

```
output int PIN_13;  
emit PIN_13(HIGH);
```

### PWM Output

For PWM output, the pin number requires the prefix `PWM_`:

```
output u8 PWM_13;  
emit PWM_13(127);
```

## Serial Communication

### Serial Communication

#### Polling Mode

A program can **await** incoming bytes from the serial as follows:

```
input byte SERIAL;  
var byte c = await SERIAL;
```

The macro `CEU_ARDUINO_SERIAL_SPEED` specifies the data transmission speed (it defaults to 9600).

In the polling mode, writing to the serial is the same as in Arduino:

<https://www.arduino.cc/en/Reference/Serial>

Note that variable and function names from Arduino must be prefixed with an underscore to be used from Céu (e.g., `_Serial.write()`).

#### Interrupt Mode

TODO

## License

## License

Céu-Arduino is distributed under the MIT license reproduced below:

Copyright (C) 2012-2017 Francisco Sant'Anna

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.