Quick start

Learn how to work with ino in few minutes.

Creating a project

Let's create a simple project:

```
$ mkdir beep
$ cd beep
$ ino init -t blink
```

Here we've created a new directory for our project and initialized project skeleton with `ino init` command. We chose to use *blink* as a project template. That will create a simple sketch for LED blinking on pin 13. Lets see what we've got:

```
$ tree
.
├── lib
└── src
    └── sketch.ino

$ cat src/sketch.ino

#define LED_PIN 13

void setup()
{
    pinMode(LED_PIN, OUTPUT);
}

void loop()
{
    digitalWrite(LED_PIN, HIGH);
    delay(100);
    digitalWrite(LED_PIN, LOW);
    delay(900);
}
```

Here we have two directories. `src` is a source directory where we can put our project's `*.[c|cpp|pde|h|hpp]` source files. `sketch.ino` was created for us, so we have a starting point. `lib` is a directory where we may put 3-rd party libraries if we would want.

Building

Lets build it:

```
$ ino build
Searching for Board description file (boards.txt) ... /usr/local/share/arduino/hardwa
Searching for Arduino core library ... /usr/local/share/arduino/hardware/arduino/core
Searching for Arduino standard libraries ... /usr/local/share/arduino/libraries
Searching for Arduino lib version file (version.txt) ... /usr/local/share/arduino/lib
Detecting Arduino software version ...  22
Searching for avr-gcc ... /usr/bin/avr-gcc
Searching for avr-g++ ... /usr/bin/avr-g++
Searching for avr-ar ... /usr/bin/avr-ar
Searching for avr-objcopy ... /usr/bin/avr-objcopy
Scanning dependencies of src
src/sketch.cpp
arduino/wiring_shift.c
arduino/wiring.c
arduino/WInterrupts.c
arduino/wiring_digital.c
arduino/wiring_pulse.c
arduino/wiring_analog.c
arduino/pins_arduino.c
arduino/HardwareSerial.cpp
arduino/WString.cpp
arduino/main.cpp
arduino/Print.cpp
arduino/WMath.cpp
arduino/Tone.cpp
Linking libcore.a
Linking firmware.elf
Converting to firmware.hex
```

Whew! A lot of work has been done behind the scenes for a single command. It's at most about finding necessary tools and directories, and compiling the standard core library. Actually you shouldn't care. The consequence is that we've got `firmware.hex` —ready to upload binary file.

Uploading

Lets upload it:

```
$ ino upload
Searching for stty ... /bin/stty
Searching for avrdude ... /usr/local/share/arduino/hardware/tools/avrdude
Searching for avrdude.conf ... /usr/local/share/arduino/hardware/tools/avrdude.conf

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.00s

avrdude: Device signature = 0x1e950f
avrdude: reading input file ".build/firmware.hex"
avrdude: writing flash (428 bytes):

Writing | ################################################## | 100% 0.08s

avrdude: 428 bytes of flash written
avrdude: verifying flash memory against .build/firmware.hex:
avrdude: load data flash data from input file .build/firmware.hex:
avrdude: input file .build/firmware.hex contains 428 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 0.06s

avrdude: verifying ...
avrdude: 428 bytes of flash verified

avrdude: safemode: Fuses OK

avrdude done.  Thank you.
```

Again, quite much output, but the job is done. Arduino flashes with its buil-in LED on pin 13.

Serial communication

OK, now lets deal with serial communication a bit. With editor of your choice change `src/sketch.ino` to:

```
void setup()
{
```

```
    Serial.begin(9600);
}

void loop()
{
    Serial.println(millis());
    delay(1000);
}
```

This should transmit number of milliseconds spent from power up every second via serial port. Lets build it:

```
$ ino build
Scanning dependencies of src
src/sketch.cpp
Linking firmware.elf
Converting to firmware.hex
```

As you can see much fewer of steps have been performed behind the scenes. It is because only things that have been changed are taken into account. This boosts up the build.

Lets upload it with `ino upload`. When uploading is done lets connect to the device with serial monitor to see what it prints:

```
$ ino serial
Searching for Serial monitor (picocom) ... /usr/bin/picocom
picocom v1.4

port is        : /dev/ttyACM0
flowcontrol    : none
baudrate is    : 9600
parity is      : none
databits are   : 8
escape is      : C-a
noinit is      : no
noreset is     : no
nolock is      : yes
send_cmd is    : ascii_xfr -s -v -l10
receive_cmd is : rz -vv

Terminal ready
0
1000
2004
```

```
3009
4014
```

That's what we want! Press Ctrl+A Ctrl+X to exit.

Tweaking parameters

All examples were done in assumption that you have Arduino Uno and it is available on default port. Now consider you have Arduino Mega 2560 and it is available on port `/dev/ttyACM1`. We have to specify this for our build steps as command-line switches.

Board model may be set with `--board-model` or `-m` switch. Port is set with `--serial-port` or `-p` switch. So lets do it:

```
$ ino build -m mega2560
$ ino upload -m mega2560 -p /dev/ttyACM1
$ ino serial -p /dev/ttyACM1
```

For the full list of board names refer to `ino build --help`.

Configuration files

It can be annoying to provide these switches over and over again. So you can save them in `ino.ini` file in project directory. Put following lines to the `ino.ini`:

```
[build]
board-model = mega2560

[upload]
board-model = mega2560
serial-port = /dev/ttyACM1

[serial]
serial-port = /dev/ttyACM1
```

Now you can build, upload and communicate via serial not having to provide any

parameters. Well, in most cases if you build for Mega 2560, you will want to upload to Mega 2560 as well. The same about serial port setting. So to don't repeat settings for different commands shared switches could be moved up to an unnamed section. So having just following lines in `ino.ini` is enough:

```
board-model = mega2560
serial-port = /dev/ttyACM1
```

Furthermore, if you have Mega 2560, it is likely that you have it for all projects you make. You can put a shared configuration file to either:

1. /etc/ino.ini
2. ~/.inorc

And it'll be used for setting default parameter values if they're not overriden by the local `ino.ini` or by explicit command-line switches.

You can provide any arguments you use on command line in a configuration file. Just specify its long name without leading `--`. E.g. `arduino-dist` but not `--arduino-dist` or `-d`.