Arduino

10.12.2014

Eclipse Arduino IDE

4.4.2016

https://www.youtube.com/watch?v=GZJ6UH_V_fg

- Download and extract Eclipse IDE for C/C++ Mars.2
- Go to Help→Eclipse Marketplace
- Install Arduino eclipse IDE V2
- Install AVR Eclipse Plugin V2.3.4
- Window → Preferences → Arduino: Point to your Arduino IDE installation (V1.6.5 recommended)

New Project

- Create a new Arduino Project
- rename your .ino file into .cpp
- add #include <Arduino.h>

Now you are ready to go!
If you want to open this project in Arduino IDE, rename the .cpp file back to .ino

Sublime IDE

https://sublime.wbond.net/packages/Arduino-like%20IDE

This should be great, since it has the sublime autocompletion, which is really missing in the original Arduiono IDE.

ino/Arturo - The Command Line Interface

10.12.2014

I just found this project on github:
https://github.com/amperka/ino

for installation on Kubuntu 14.04 we need:

```
cd ~/programms
git clone git://github.com/amperka/ino.git
cd ino
sudo apt-get install python-pip python-jinja2
sudo pip install glob2 jinja2 pyserial configobj ordereddict argparse
sudo apt-get install python-serial
sudo make install
```

Here is the documentation for it:
http://inotool.org/quickstart

Update: ino forked into Arturo

since ino wasn't supported any more, a fork of the project named „Arturo“ was created:
https://github.com/scottdarch/Arturo

~/.inorc

in this config file, which is not installed by default, we can set default settings. For me the most important one is the path to the arduino distribtion.

.inorc

```
[build]
board-model = nano328
arduino-dist = /home/karl/Programme/arduino-1.0.6/

[upload]
board-model = nano328
arduino-dist = /home/karl/Programme/arduino-1.0.6/
serial-port = /dev/ttyUSB0

[serial]
serial-port = /dev/ttyUSB0
```

Important: Don't forget to define the arduino-dist for both, the build and the upload command. Otherwise the upload command doesn't find the correct build-path, which as a suffix which is generated from the arduino-dist-dir.

Error when moving host platform

Today I moved my ino-project to a new computer, and forgot to erase the .build directory. I got nothing-saying error messages like this:

```
karl@LackerServer:/xxxxxxxxxxxxxxx/yyyyyyyyyy$ ino build
```

```
Traceback (most recent call last):
  File "/usr/local/bin/ino", line 6, in <module>
    main()
  File "/usr/local/lib/python2.7/dist-packages/ino/runner.py", line 76, in
main
    args.func(args)
  File "/usr/local/lib/python2.7/dist-packages/ino/commands/build.py", line
286, in run
    self.make('Makefile.sketch')
  File "/usr/local/lib/python2.7/dist-packages/ino/commands/build.py", line
216, in make
    ret = subprocess.call([self.e.make, '-f', makefile, 'all'])
  File "/usr/lib/python2.7/subprocess.py", line 493, in call
    return Popen(*popenargs, **kwargs).wait()
  File "/usr/lib/python2.7/subprocess.py", line 679, in __init__
    errread, errwrite)
  File "/usr/lib/python2.7/subprocess.py", line 1249, in _execute_child
    raise child_exception
OSError: [Errno 2] No such file or directory
```

Compile existing project

If we have an existing folder, with it's source files, we have to do a little adoption, so we can compile it with **ino**:
We need a defined folder structure:

```
projectpath/
  ./lib/
  ./src/
    ./sketch.ino
```

So the starting point of the project must be in sketch.ino.
Then we can run this:

```
:~/projectpath$ ino build
```

For uploading to the board we use

```
ino upload
```

Example

Her is an example file structure after building the project

```
.
└── simple-control
    ├── .build
    │   ├── environment.pickle
```

```
    │   ├── nano328-a89b84a3
    │   │   ├── arduino
    │   │   │   ├── avr-libc
    │   │   │   │   ├── malloc.d
    │   │   │   │   ├── malloc.o
    │   │   │   │   ├── realloc.d
    │   │   │   │   └── realloc.o
    │   │   │   ├── CDC.d
    │   │   │   ├── CDC.o
    │   │   │   ├── dependencies.d
    │   │   │   ├── HardwareSerial.d
    │   │   │   ├── HardwareSerial.o
    │   │   │   ├── HID.d
    │   │   │   ├── HID.o
    │   │   │   ├── IPAddress.d
    │   │   │   ├── IPAddress.o
    │   │   │   ├── libarduino.a
    │   │   │   ├── main.d
    │   │   │   ├── main.o
    │   │   │   ├── new.d
    │   │   │   ├── new.o
    │   │   │   ├── Print.d
    │   │   │   ├── Print.o
    │   │   │   ├── Stream.d
    │   │   │   ├── Stream.o
    │   │   │   ├── Tone.d
    │   │   │   ├── Tone.o
    │   │   │   ├── USBCore.d
    │   │   │   ├── USBCore.o
    │   │   │   ├── WInterrupts.d
    │   │   │   ├── WInterrupts.o
    │   │   │   ├── wiring_analog.d
    │   │   │   ├── wiring_analog.o
    │   │   │   ├── wiring.d
    │   │   │   ├── wiring_digital.d
    │   │   │   ├── wiring_digital.o
    │   │   │   ├── wiring.o
    │   │   │   ├── wiring_pulse.d
    │   │   │   ├── wiring_pulse.o
    │   │   │   ├── wiring_shift.d
    │   │   │   ├── wiring_shift.o
    │   │   │   ├── WMath.d
    │   │   │   ├── WMath.o
    │   │   │   ├── WString.d
    │   │   │   └── WString.o
    │   │   ├── firmware.elf
    │   │   ├── firmware.hex
    │   │   ├── Makefile
    │   │   ├── Makefile.deps
    │   │   ├── Makefile.sketch
    │   │   └── src
```

```
    │   │       ├── dependencies.d
    │   │       ├── message-interpreter.cpp
    │   │       ├── message-interpreter.d
    │   │       ├── message-interpreter.o
    │   │       ├── sketch.cpp
    │   │       ├── sketch.d
    │   │       └── sketch.o
    │   └── uno-a89b84a3
    ├── lib
    └── src
        ├── message-interpreter.h
        ├── message-interpreter.ino
        ├── simple-control.h
        └── sketch.ino
```

avrdude and AVR ISP mkII with USB on Linux

unlike as decribed here, which is a bit more complicated http://stackoverflow.com/a/5414566 we just add the permissions for the USB device similar to that of an Android decivce in Debug-Mode:

create a file /etc/udev/rules.d/51-avrisp.rules with

```
sudo kate /etc/udev/rules.d/51-avrisp.rules
```

enter this content:

```
# AVR ISP mkII (like an Android Device in Debug-Mode):
SUBSYSTEM=="usb", ATTR{idVendor}=="03eb", MODE="0666", GROUP="dialout"
```

save an close this file.

restart udev:

```
sudo service udev restart
```

unplug the programmer and plug it in again. now avrdude can be used without sodo!

an example call to avrdude could then be:

```
avrdude -U flash:w:firmware.hex -p m328p -c avrispmkII -P usb
```

Fuse Bit Calculator

http://www.engbedded.com/fusecalc/

avrdude

Example Invocations

http://www.nongnu.org/avrdude/user-manual/avrdude_6.html

Arduino and size optimization

- http://blog.oscarliang.net/check-ram-memory-usage-arduino-optimization/ *
- http://www.avrfreaks.net/forum/how-optimize-size
- http://www.avrfreaks.net/forum/tut-c-gcc-and-progmem-attribute?name=PNphpBB2&file=viewtopic&t=38003
- very interesting: PROGMEM and strings in flash with

  ```
  PSTR("My String in flash")
  ```

  access this string with

  ```
  pgm_read_byte_near(str+i)
  ```

  , where str is the pointer given by a PSTR()-expression.

- http://playground.arduino.cc/Main/CorruptArrayVariablesAndMemory

Arduino IDE for the ESP8266

https://github.com/esp8266/Arduino

C++ new operator

Here is a great article about this topic.

http://arduino.land/FAQ/content/4/20/en/can-i-use-new-and-delete-with-arduino.html
http://www.cplusplus.com/reference/new/operator%20new/

Most useful is the placement operator for static allocated memory.

```
#include <new> // very important, otherwise you get helpless compile errors!
#include "memorydispenser.h"
```

```
#define m(x) new (MemoryDispenser::get(sizeof(x))) x


// usage example:
JobController job = m(JobController(2));
```

software, english, collection