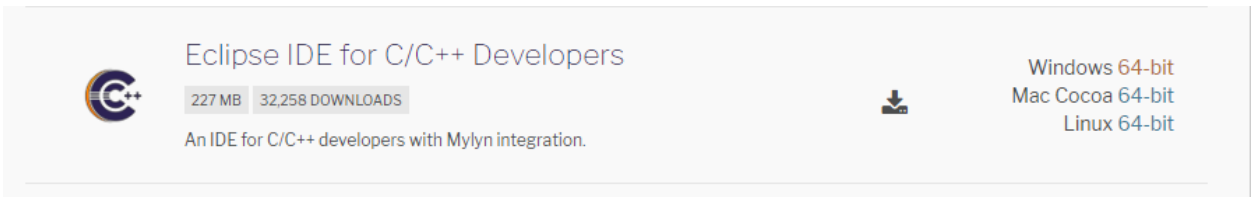
Download von Eclipse von https://www.eclipse.org/downloads/

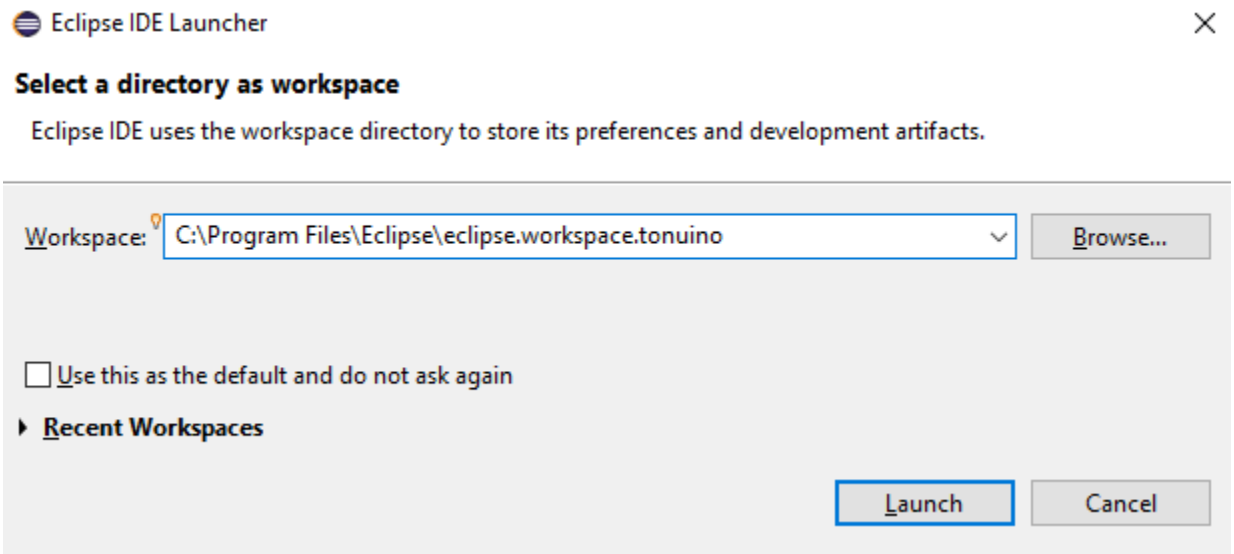
“Download Packages” (rot umrandet) wählen (Nicht den Installer herunterladen!)

Aus der erscheinenden Auswahlliste “Eclipse IDE for C/C++ Developers“ wählen für Euer OS.
Für mich: Win (10), 64-bit:

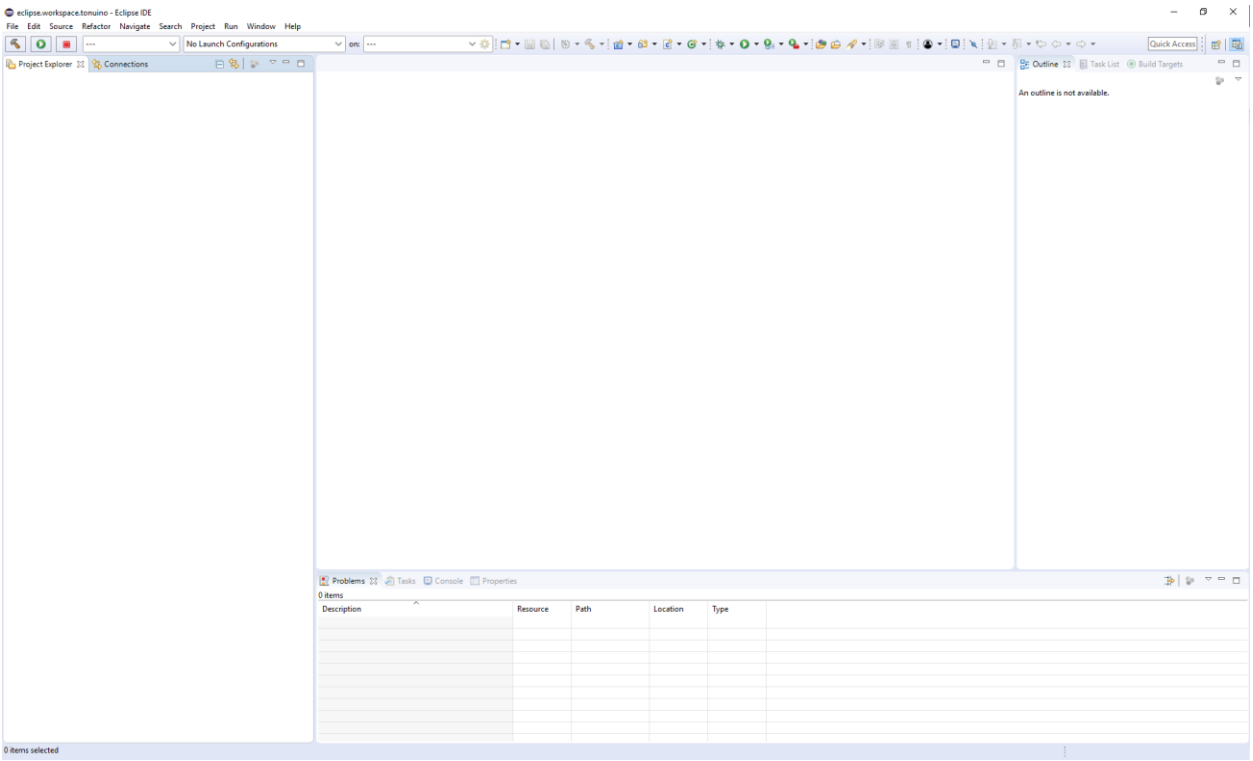


Der Download ist ein ZIP Archiv, das einfach in einen leeren Ordner entpacken. Im erzeugten Ordner „eclipse“ die „eclipse.exe“ starten.

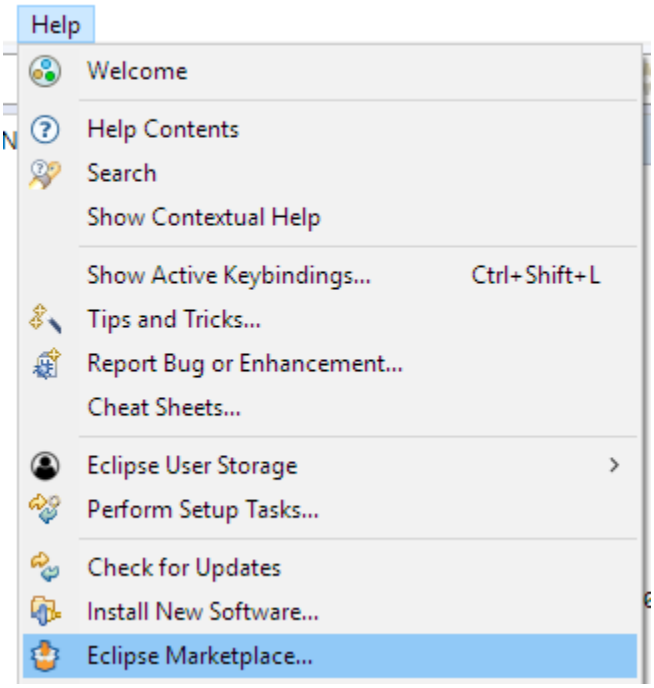
Beim ersten Start kann ein Workspace gewählt bzw. angelegt werden. Hierfür ein leeres Verzeichnis angeben, dieses wird erzeugt falls nicht vorhanden.



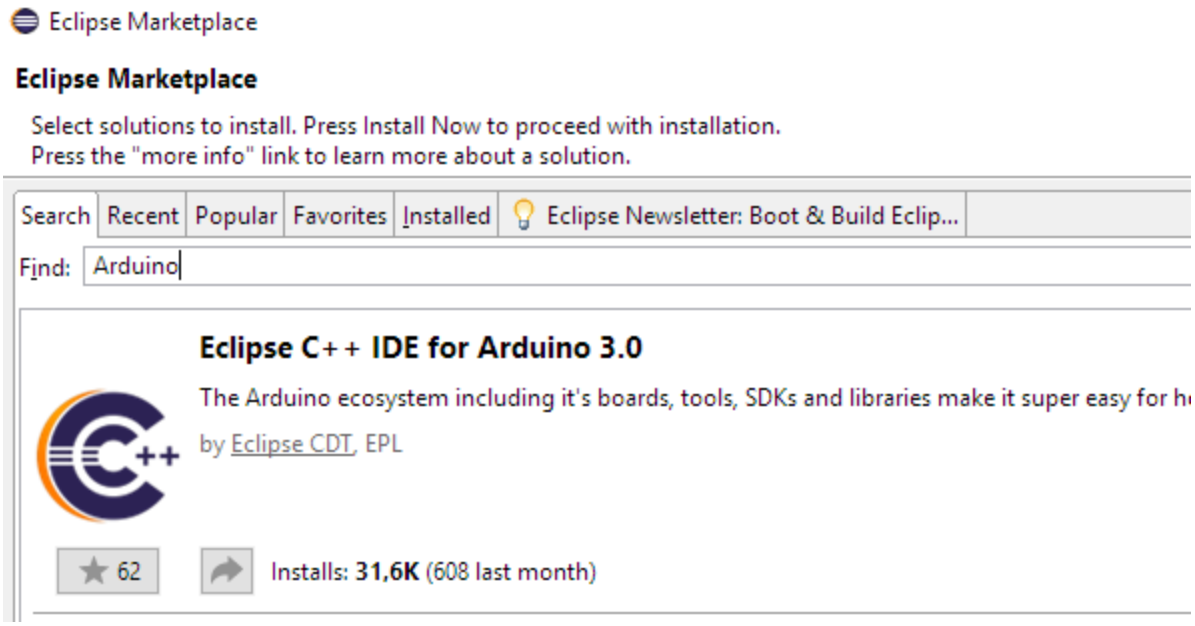
Es erscheint die Eclipse IDE. Das Welcome Fenster schließen (Kreuz im Tabreiter, erscheint bei jedem Start neu, außer man wählt es mit der Checkbox ab, also kann man das auch später erkunden…). Dann sollte Eclipse so aussehen:



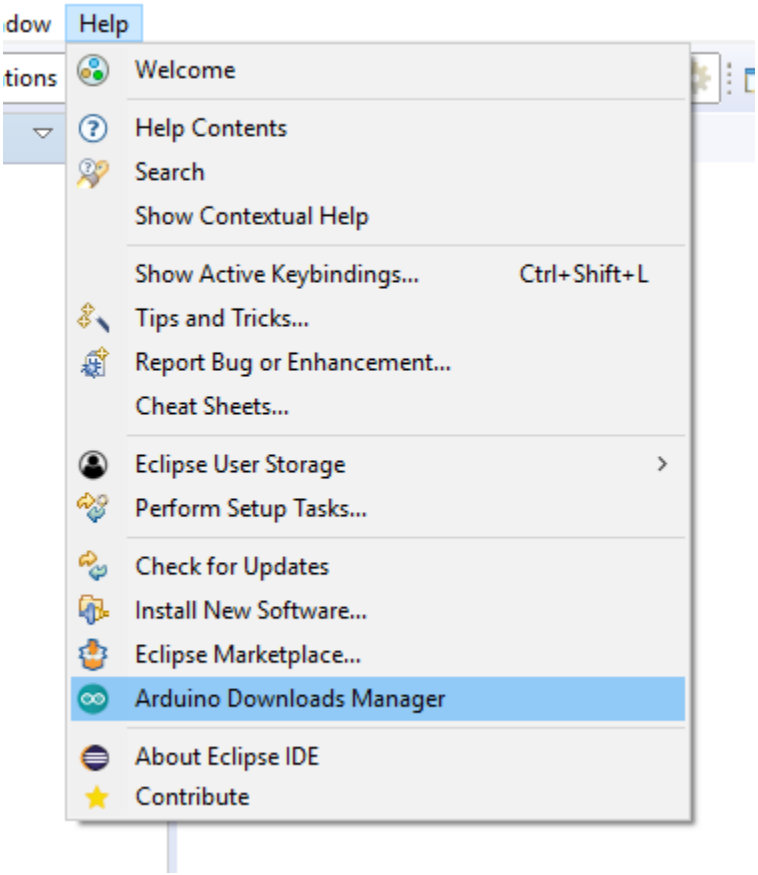
Zuerst muss eventuell noch die Arduino IDE über den Eclipse Marketplace hinzugefügt werden, falls im Help Menü unter „Eclipse Marketplace“ NICHT der „Arduino Downloads Manager“ erscheint:



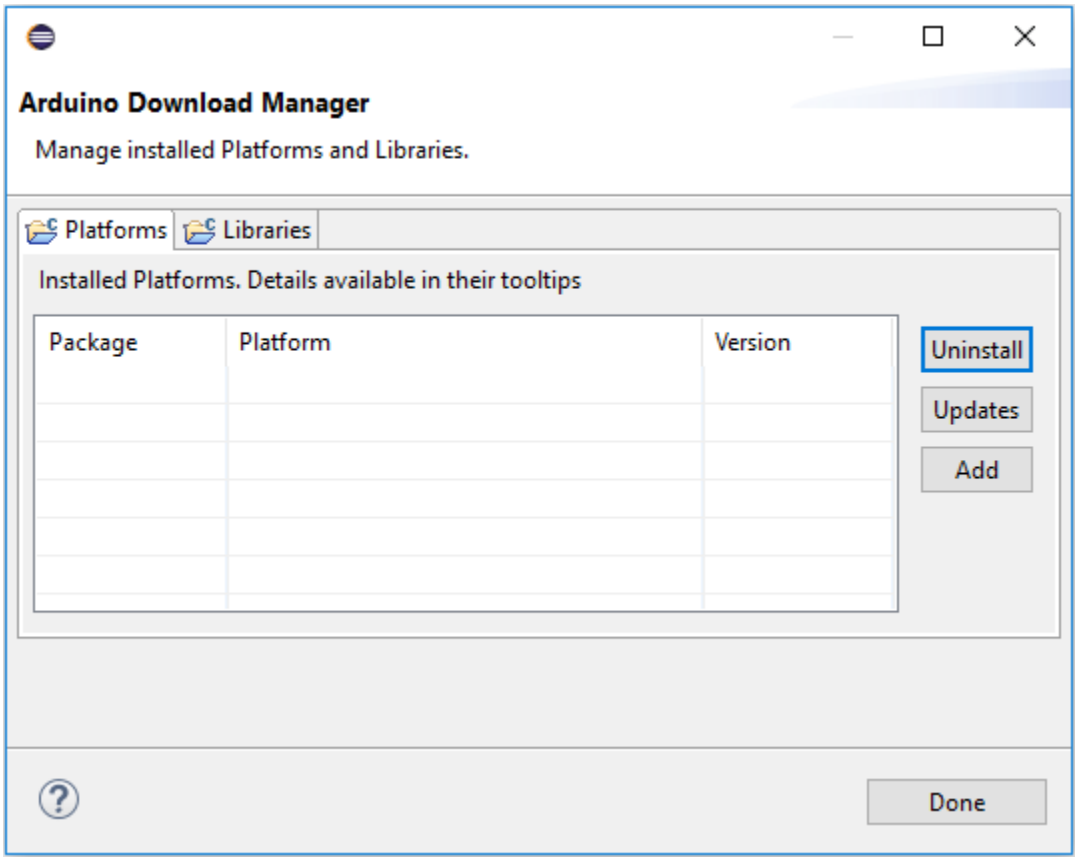
Öffnen Marketplace und suchen nach Arduino. Installieren des Eintrags mit „Install“ (rechts im Eintrag, nicht im Bild zu sehen)



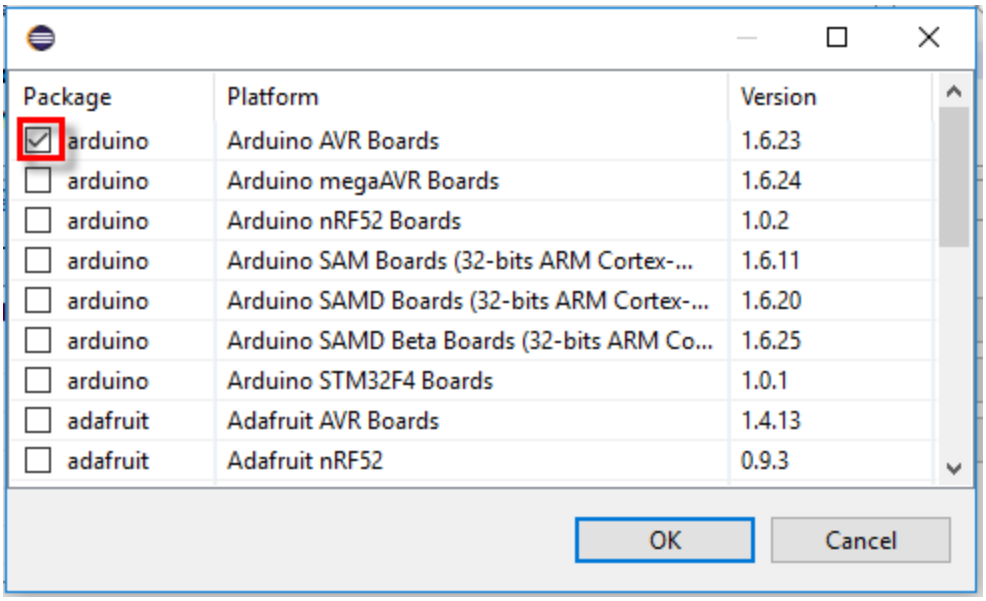
Dann müssen die Compiler/Toolchain etc. für das Arduino Board sowie die im Projekt genutzten Bibliotheken heruntergeladen werden. Hierzu den „Arduino Downloads Manager“ im Help Menü nutzen:



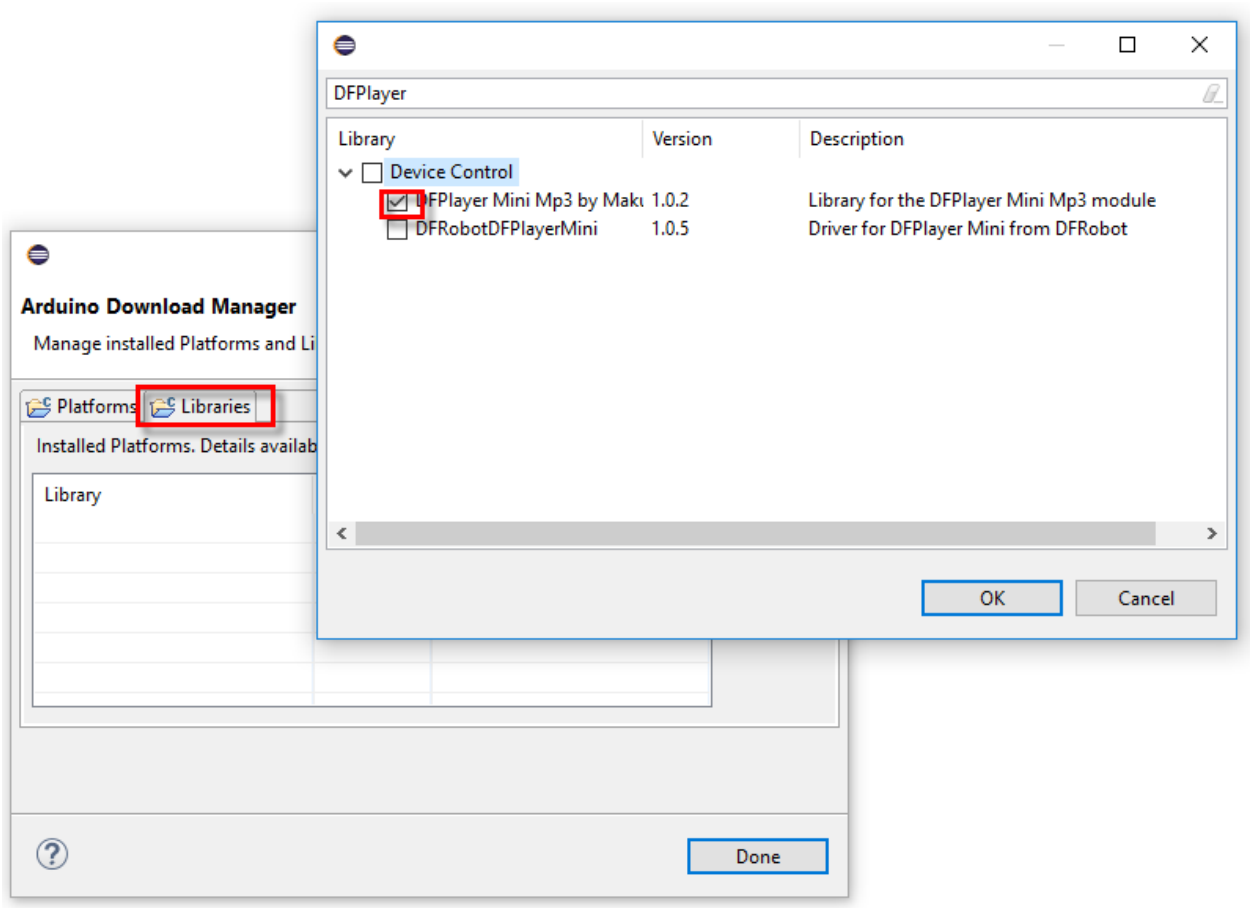
Zuerst die Arduino Plattform mit Compiler und Tools hinzufügen



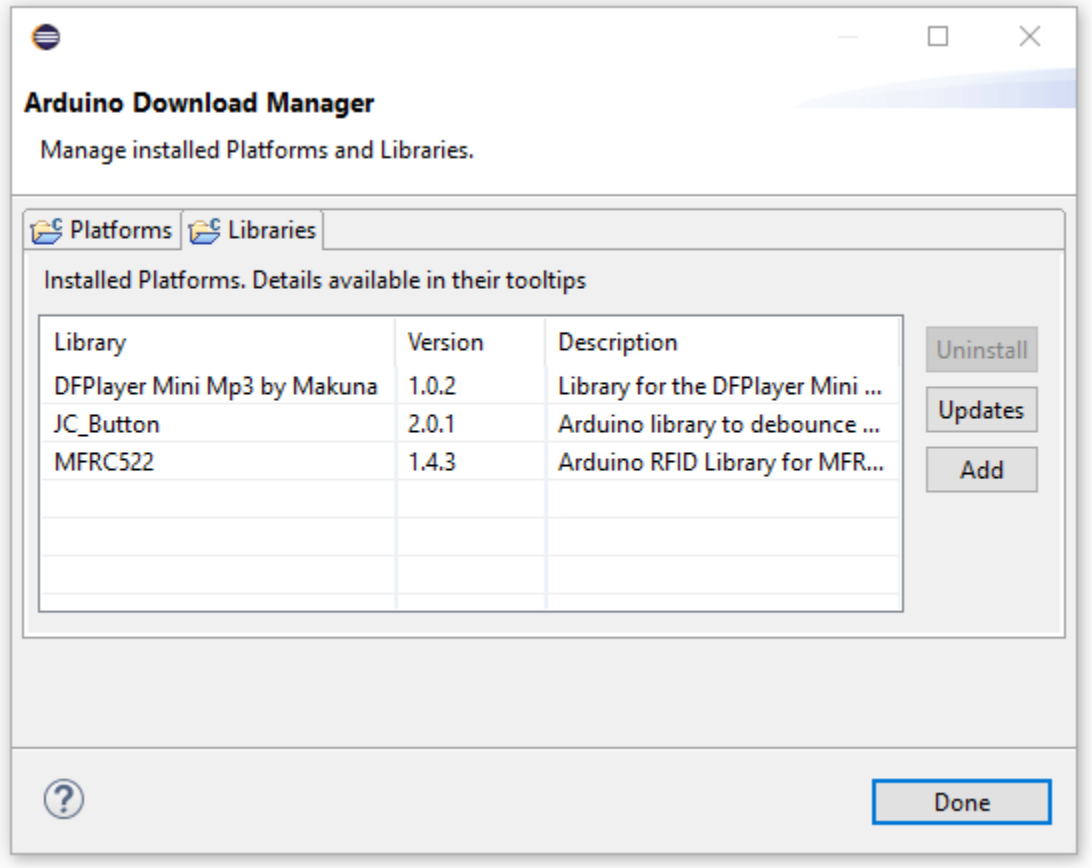
Der Nano ist im Paket „Arduino AVR Boards“



Dann auf den Reiter Libraries wechseln und die 3 Bibliotheken hinzufügen

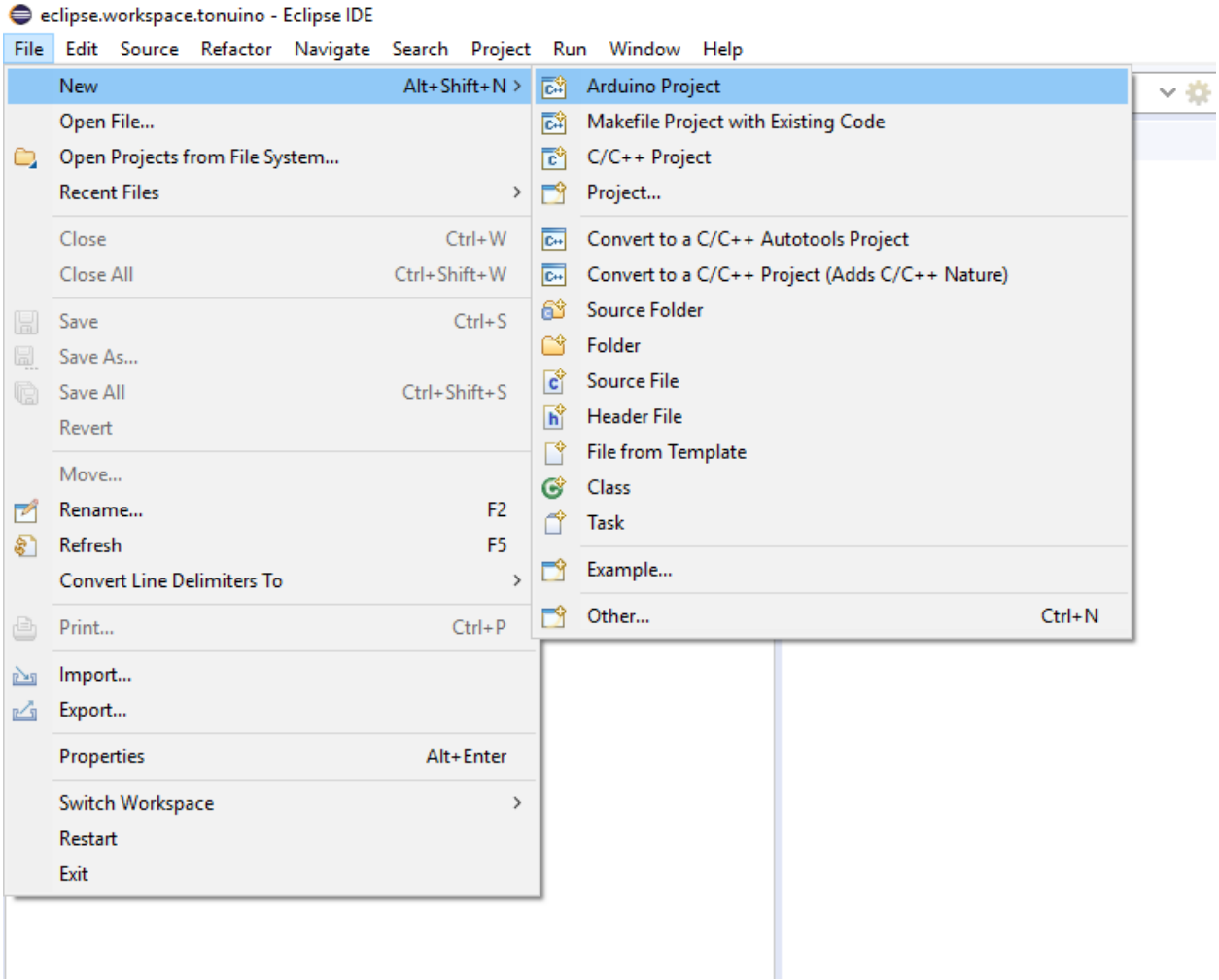


Danach sollte es so aussehen:

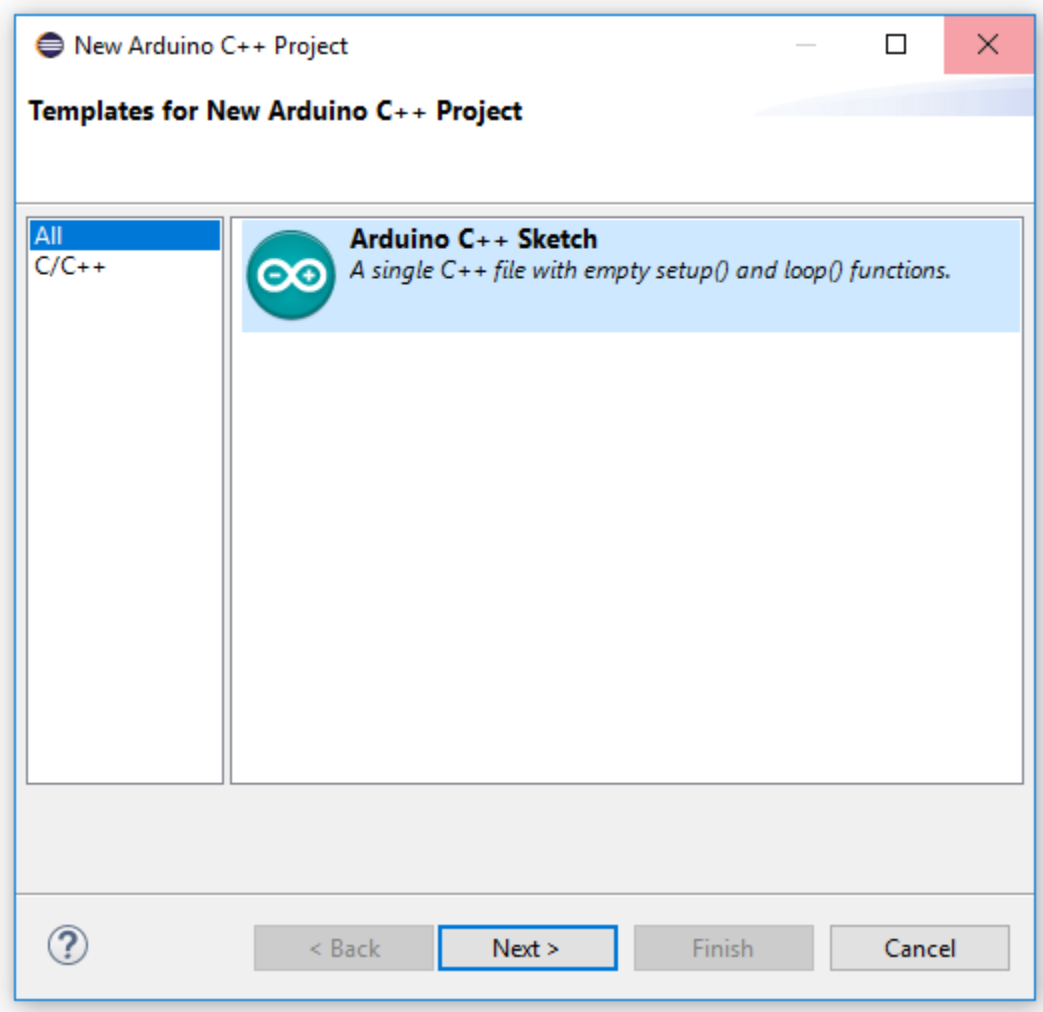


Den Download Manager mit Done beenden.

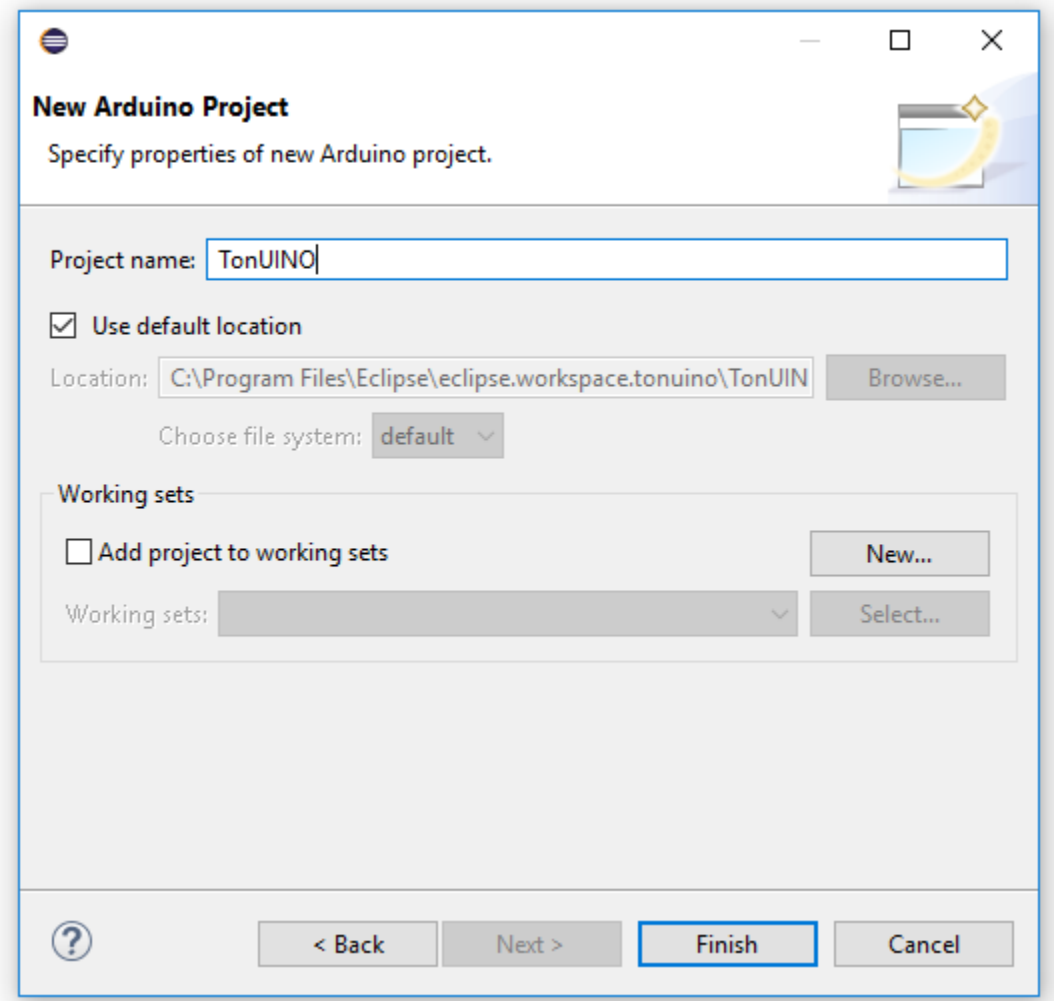
Dann ein neues Arduino Projekt erzeugen:



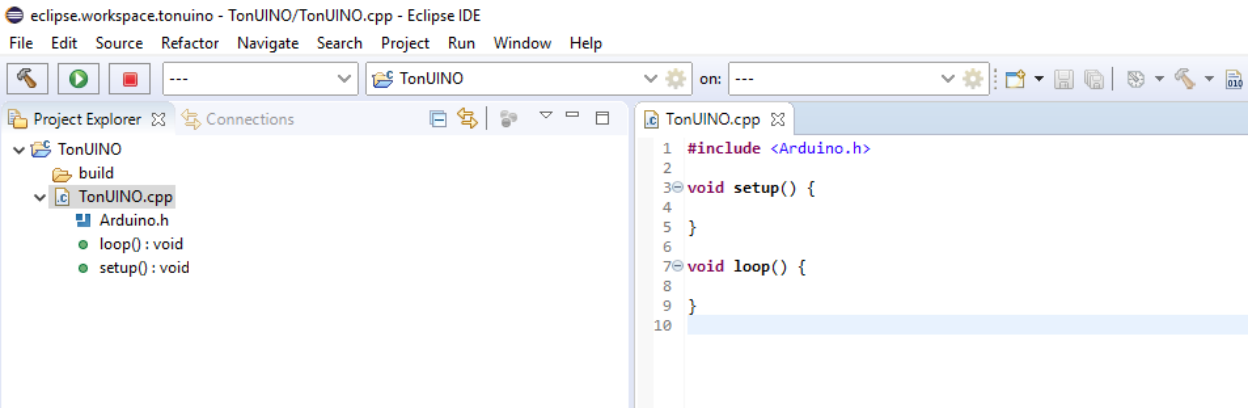
Es gibt nur eine Vorlage, mit Next bestätigen:



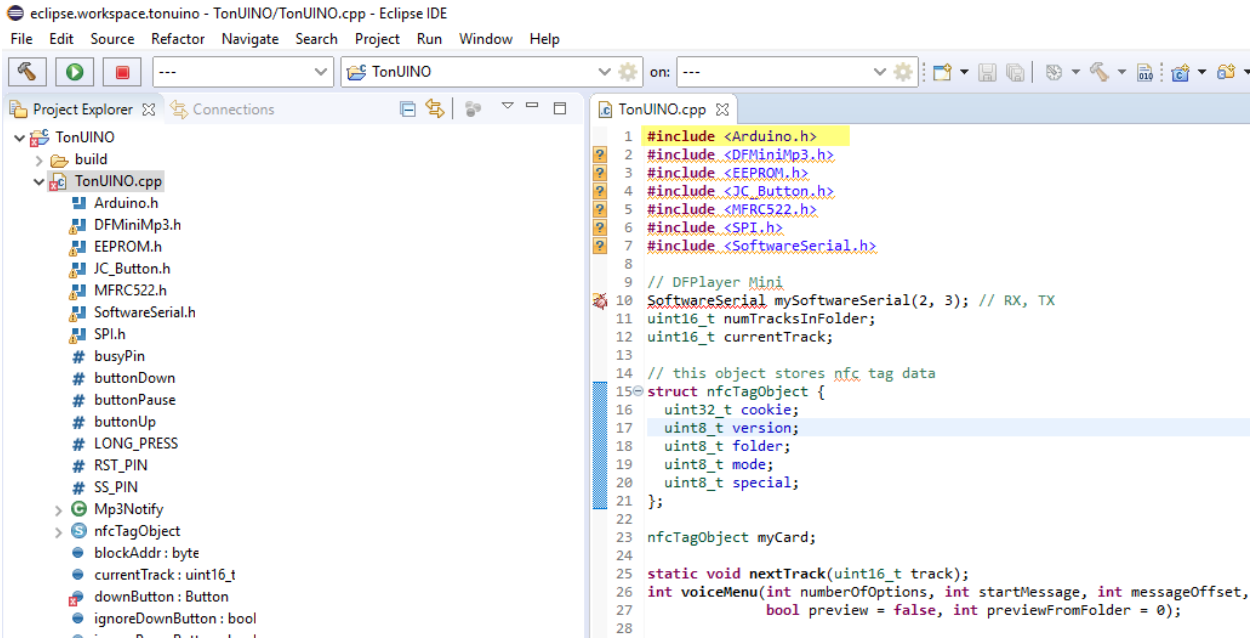
Projektname festlegen und Speicherort wählen



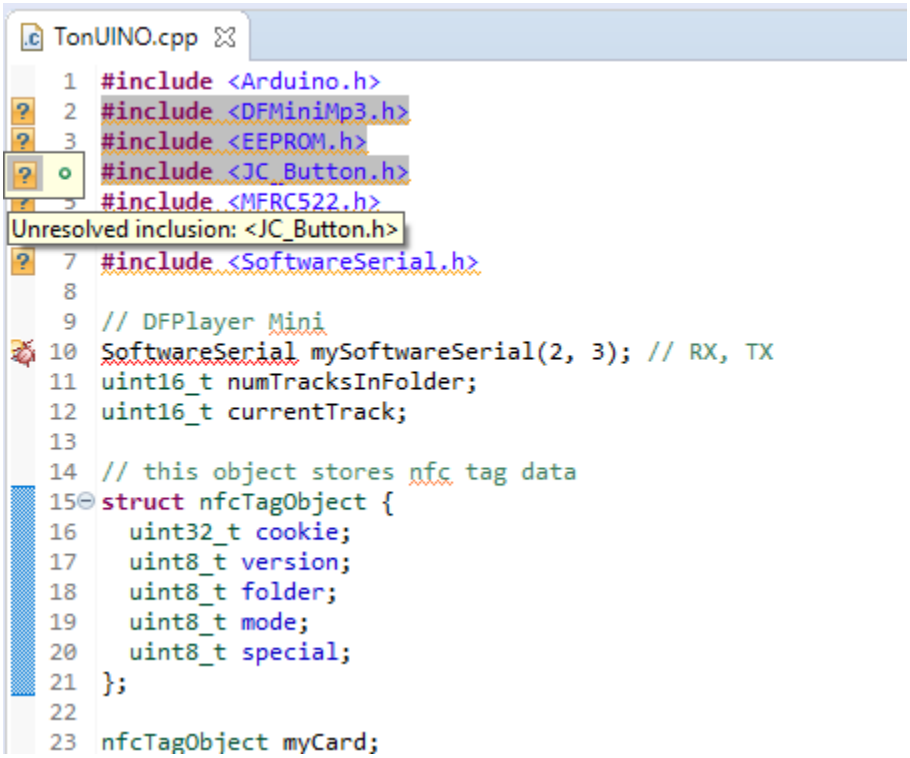
Eclipse erzeugt ein Projekt mit Haupdatei „TonUINO.cpp“:



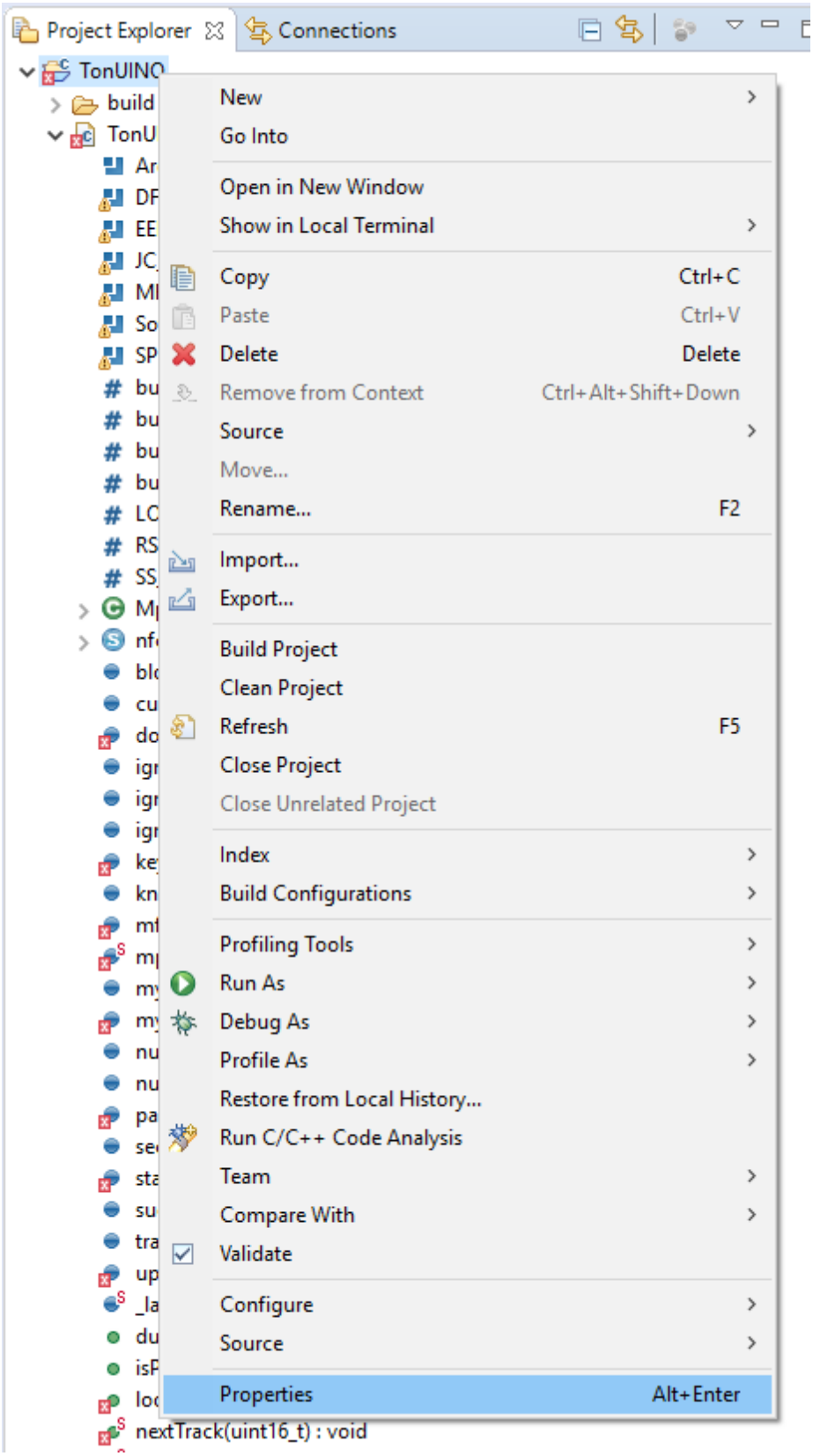
Löschen der Rümpfe von setup und loop Methode, #include <Arduino.h> muss erhalten bleiben. Dann Einfügen des Codes aus der INO Datei.



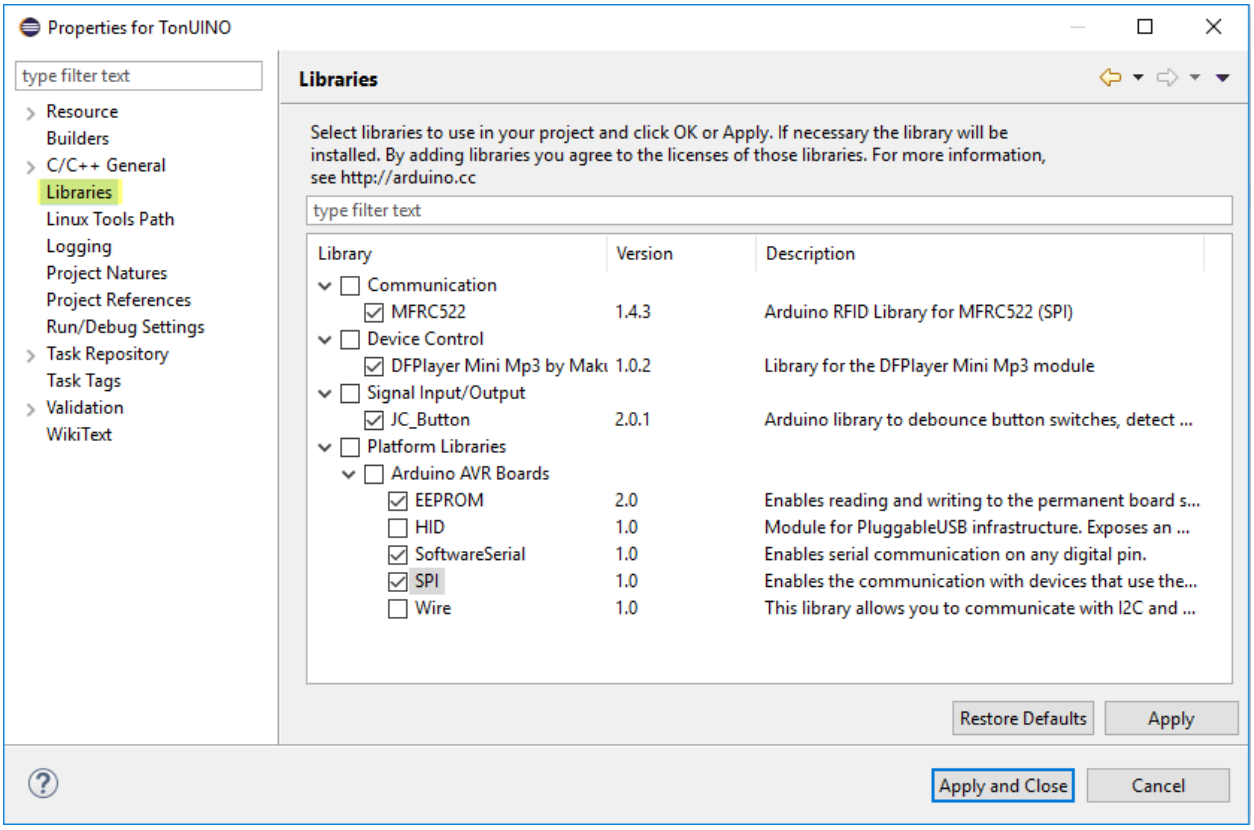
Danach gibt es noch jede Menge Fehler vom Compiler (bzw. der Codeprüfung, wirklich compiliert ist eigentlich noch nichts)



Dem neuen Projekt fehlen noch die genutzten Bibliotheken. Hinzufügen der Bibliotheken zum neuen Projekt mittels Rechtsklick auf das TonUINO Projekt -> Properties:

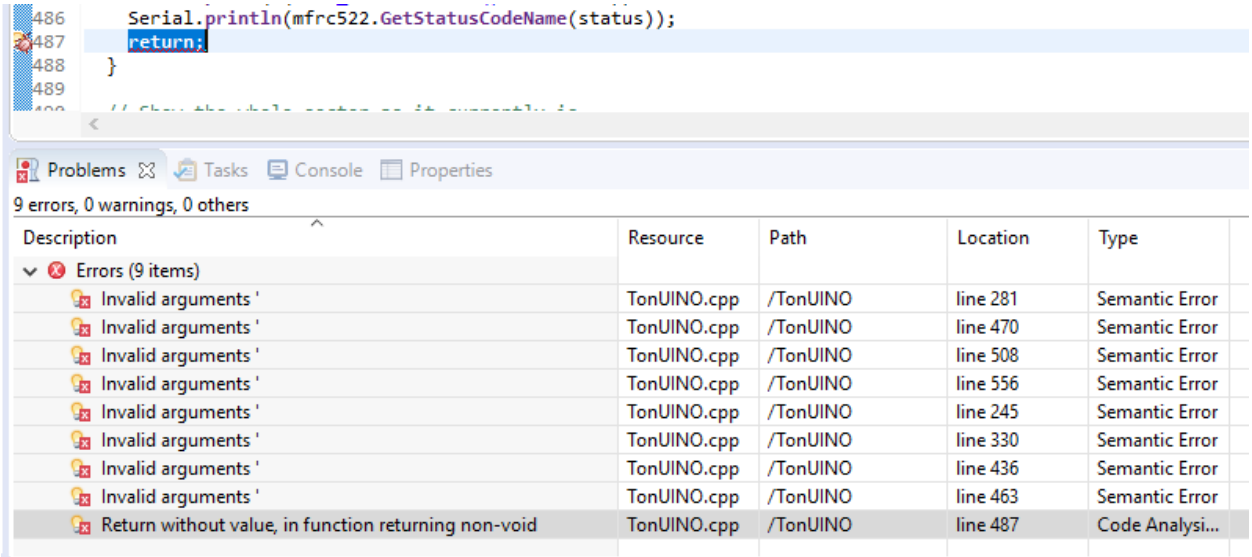


Im Abschnitt Libraries: Auswahl der benötigten Bibliotheken:



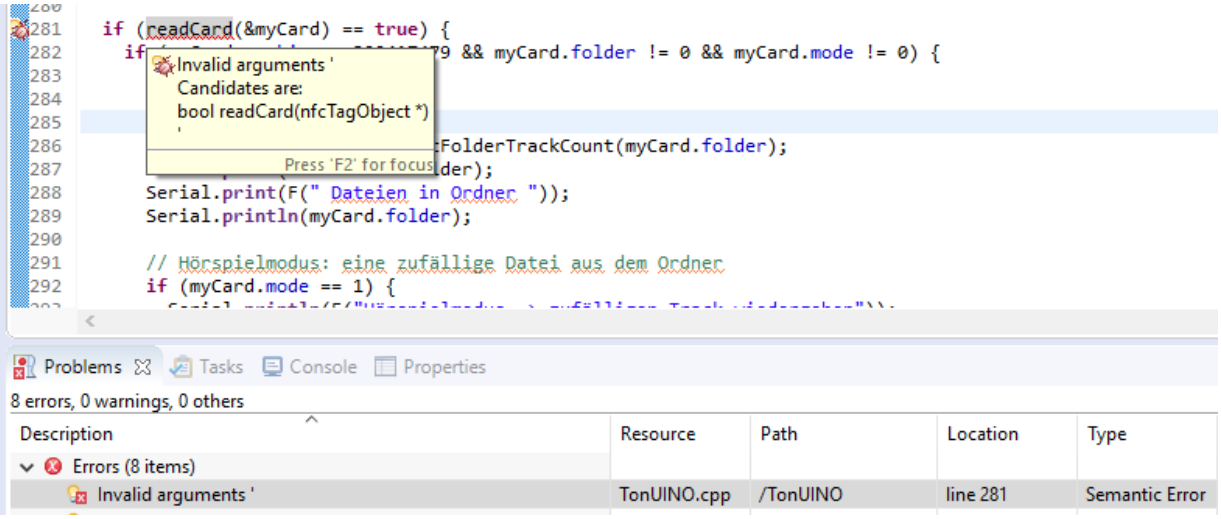
Bestätigen mit Apply and Close, die meisten Fehler verschwinden. Es verbleiben ggf. – versionsabhängig von der INO Datei – Fehler, die die Arduino IDE akzeptiert aber ein CPP Compiler nicht. Diese ggf. beseitigen, am Beispiel Master Branch vom 11.11.2018:

a) Fehlender Rückgabewert in Methode mit Rückgabewert bool:



⇨ Korrektur mit „`return returnValue;`“;

b) Semantische Fehler aufgrund fehlender Methoden-Prototypen (üblicherweise in den .h Headerfiles, gibt es bei INO Dateien nicht, da wird das wohl toleriert von der Arduino IDE)



⇨ Korrektur mit Einfügen der Zeilen

```
bool readCard(nfcTagObject *nfcTag);
void dump_byte_array(byte *buffer, byte bufferSize);
void resetCard();
void setupCard();
void writeCard(nfcTagObject nfcTag);
```

hinter der Zeile

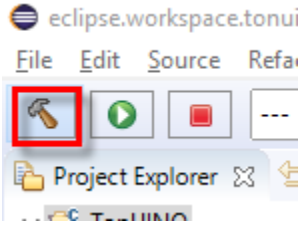
```
int voiceMenu(int numberOfOptions, int startMessage, int messageOffset,
              bool preview = false, int previewFromFolder = 0);
```

→ Ergebnis

```
25 static void nextTrack(uint16_t track);
26 int voiceMenu(int numberOfOptions, int startMessage, int messageOffset,
27              bool preview = false, int previewFromFolder = 0);
28 bool readCard(nfcTagObject *nfcTag);
29 void dump_byte_array(byte *buffer, byte bufferSize);
30 void resetCard();
31 void setupCard();
32 void writeCard(nfcTagObject nfcTag);
33 bool knownCard = false;
34
```

Ergebnis von a) und b): keine Warn-/Fehlermeldungen mehr:

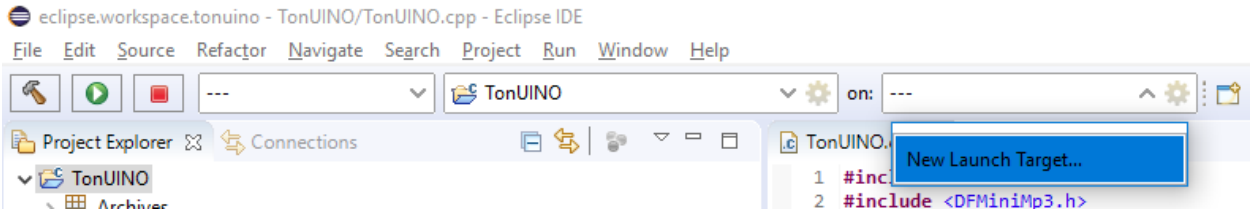
Nun kann „richtig“ compiliert werden, hierfür den Hammer wählen:



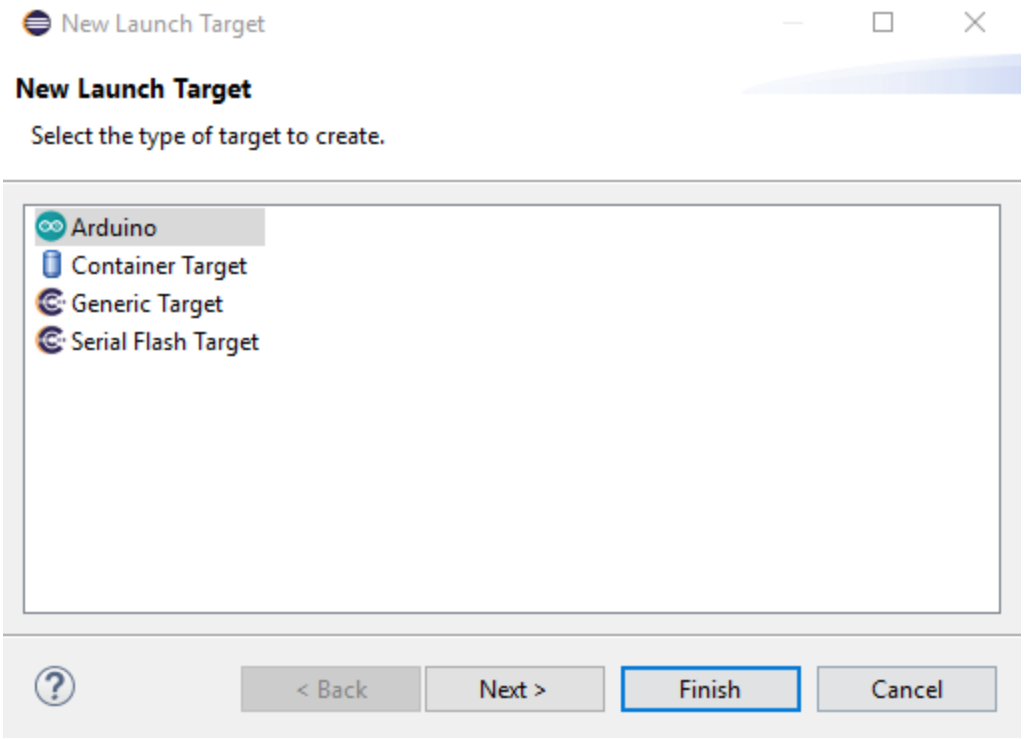
In der Console View erscheint jede Menge Output vom Compiler..:

```
…
…
…packages/arduino/tools/avr-gcc/5.4.0-atmel3.6.1-arduino2/bin/avr-objcopy" -O ihex -R
.eeprom  "./TonUINO.elf" "./TonUINO.hex"
"…packages/arduino/tools/avr-gcc/5.4.0-atmel3.6.1-arduino2/bin/avr-objcopy" -O ihex -
j .eeprom --set-section-flags=.eeprom=alloc,load --no-change-warnings --change-
section-lma .eeprom=0  "./TonUINO.elf" "./TonUINO.eep"
Program store usage: 15159 of maximum 32256 bytes
Initial RAM usage: 517 of maximum 2048 bytes
```

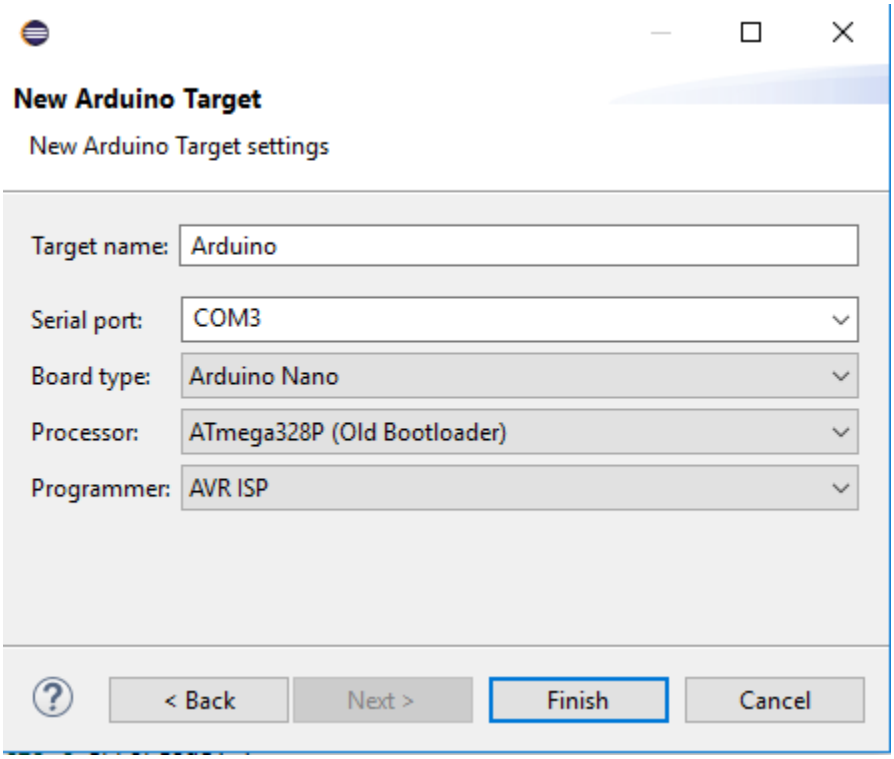
Zum Übertragen auf den Arduino fehlt nun nur noch das Launch Target, zum erzeugen „New Launch Target…“ wählen:



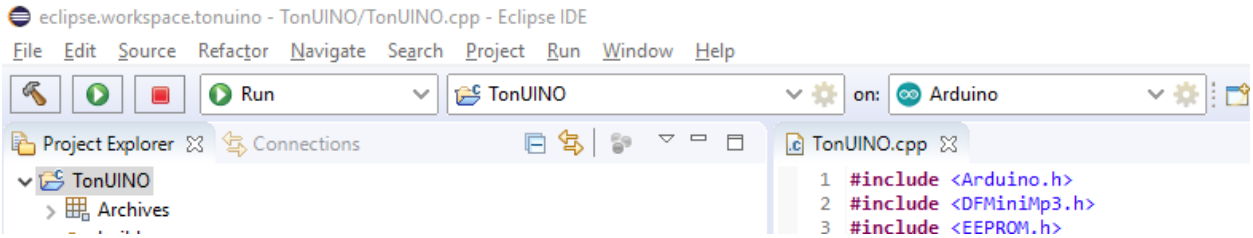
„Arduino“ als Target wählen, mit Next bestätigen:



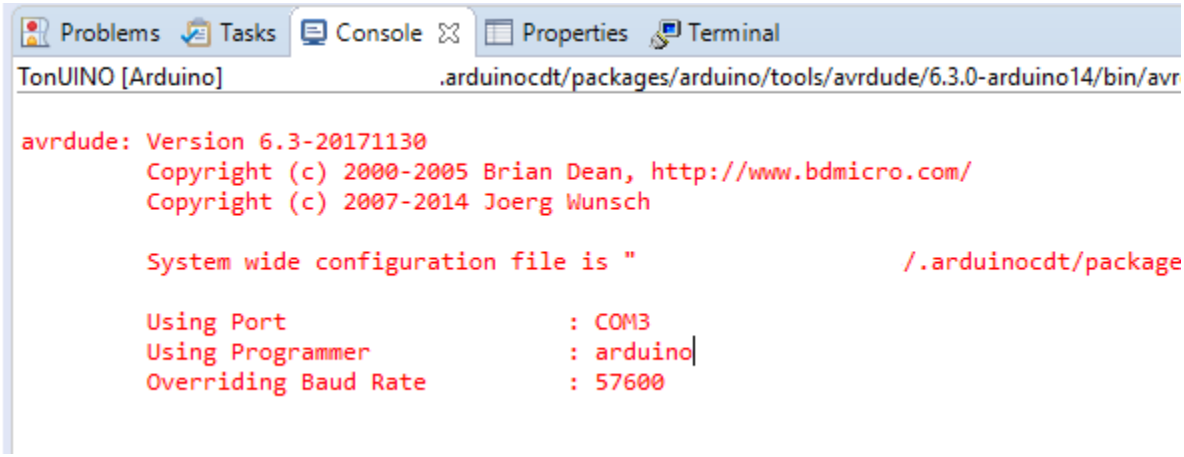
Target Name vergeben, Serial Port wählen und die Einstellungen entsprechend des Boards festlegen. Mit Finish bestätigen:



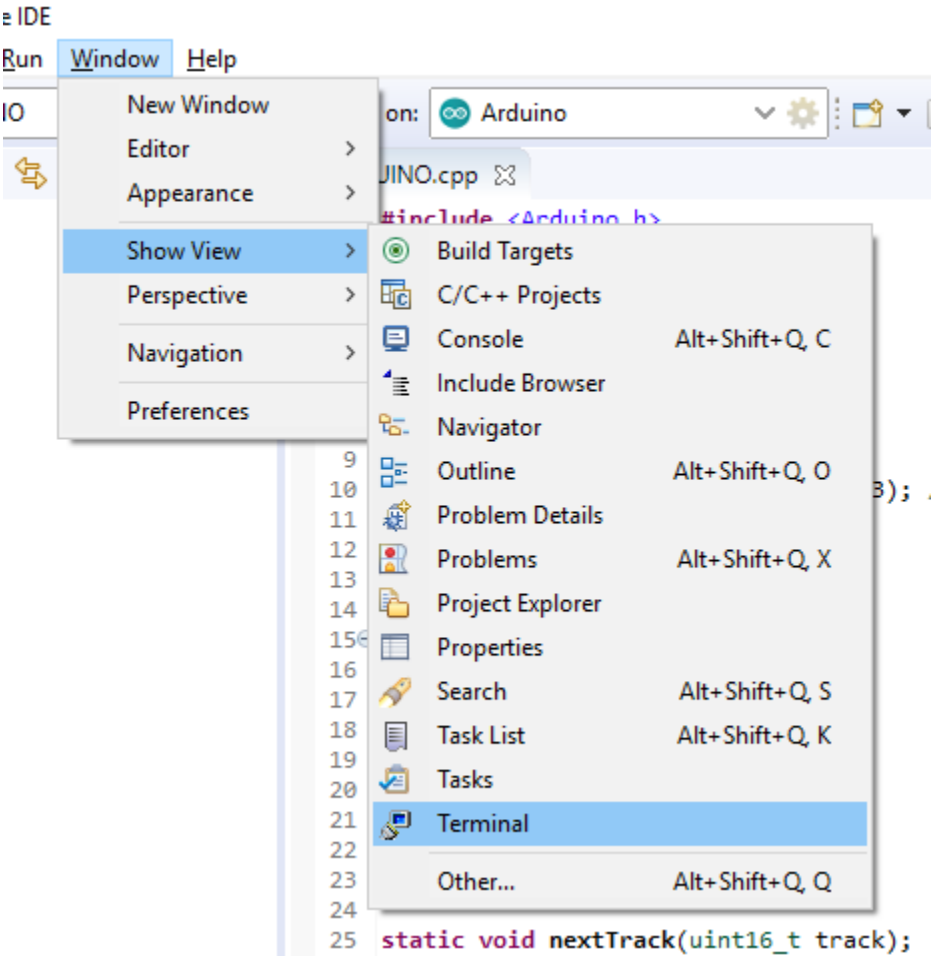
Danach sollte es so aussehen:



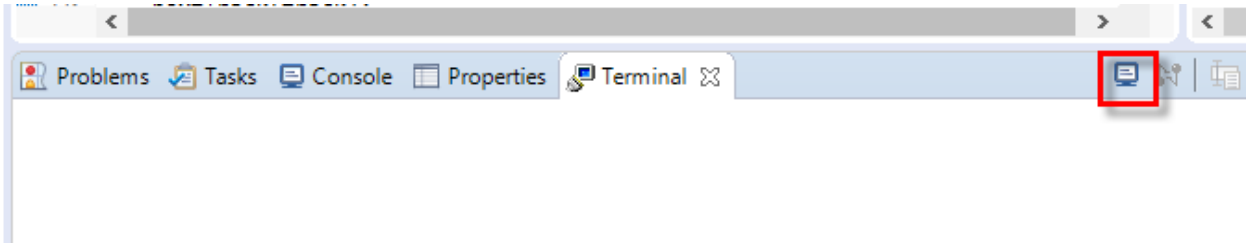
Mittels des Buttons neben dem Hammer kann nun die Übertragung auf den Arduino erfolgen – fertig. Ggf. erfolgt hierfür noch einmal ein automatisches Build durch den Compiler.



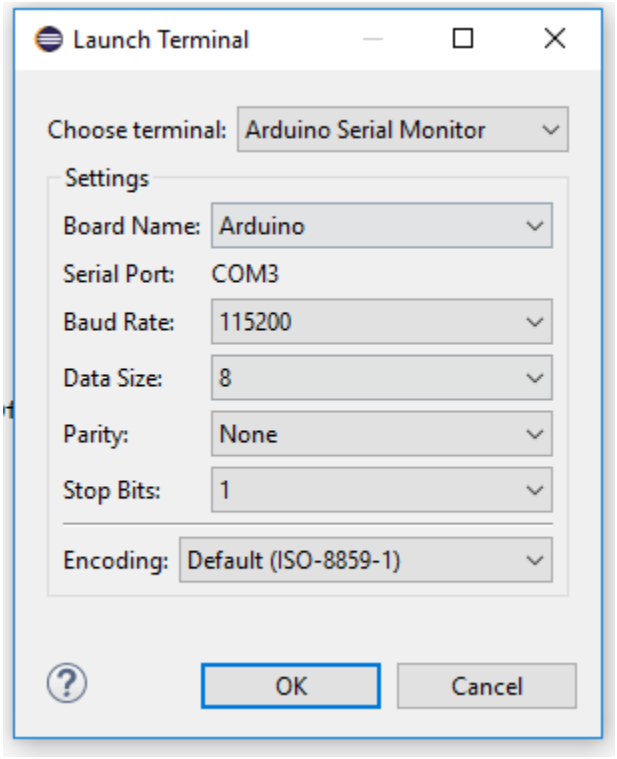
Einen „Serial Monitor“ gibt’s natürlich auch – über Window -> Terminal die View hierfür öffnen bzw. der IDE hinzufügen:



In der View über den „Monitor“ den Serial Monitor aktivieren:



Im erscheinenden Popup Einstellungen prüfen und ggf. korrigieren:



Es erscheint die Konsole: