Weather - Station

Version - 0.1.0

Moser Martin

Table of Contents

Introduction

Disclaimer

Arduino IDE

At the time this document was written, the Arduino IDE with version 1.8.13 was used. If another version is used, the IDE may looks not the same or possible some steps are not one by one reproducible. But the IDE is in such a state that it does not change to much from version to version so it should not be a problem do follow along.

Libraries

Kotlinx Serialization

TornadoFX

JSerialComm

Shortcuts

Table 1. Shortcuts

Shortcut	Action
`Ctrl + C`	Connect to Arduino

Setup

Arduino IDE

Used version at the time of writing Arduino SAMD Boards package = version 1.8.9 Arduino_MKRENV = version 1.1.0.

MKR board

To be able to program the Arduino using the official Arduino IDE, the board has to be installed first. This could be done with the integrated board manager. Select 'Tools' → 'Board' → 'Boards Manager...''. This will open up a new window which provide functionality to add, update or remove boards. Search for a package called 'Arduino SAMD Boards (32-bits ARM Cortex M0+)'. This package is required for the Arduino MKR 1010 WiFi. Select the newest version of the library and install it.

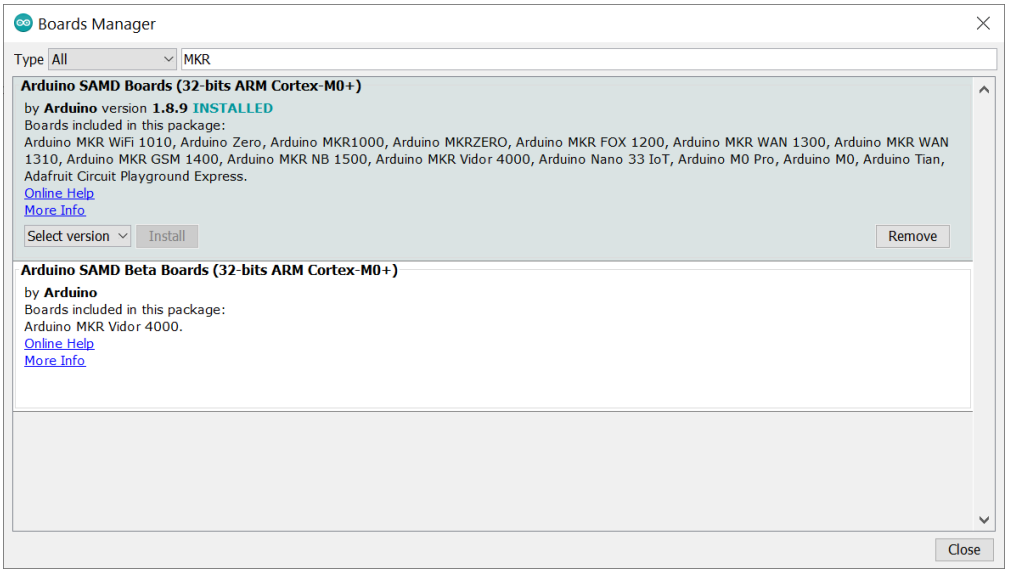


Figure 1. Arduino IDE Board Manager

After installation, the board must be selected. This is required, for the IDE to know for which controller the code must be compiled. Select 'Tools' → 'Board' → 'Arduino SAMD Boards (32-bits ARM Cortex M0+)' → 'Arduino MKR WiFi 1010'

MKR Env shield

The usage of the Env shield requires a library which provides the functions to read the sensors. It is an official Arduino library, so the installation could be done using the Arduino IDE. Select 'Tools' → 'Manage Libraries...'. This will open the Library Manager, where libraries could be installed, updated or uninstalled. Search for a library called 'Arduino_MKRENV' and install the newest version.

It is not required necessary to use the library but it is really recommended since it already implement all functions. If the library is not used all of the functions must be implemented by hand.

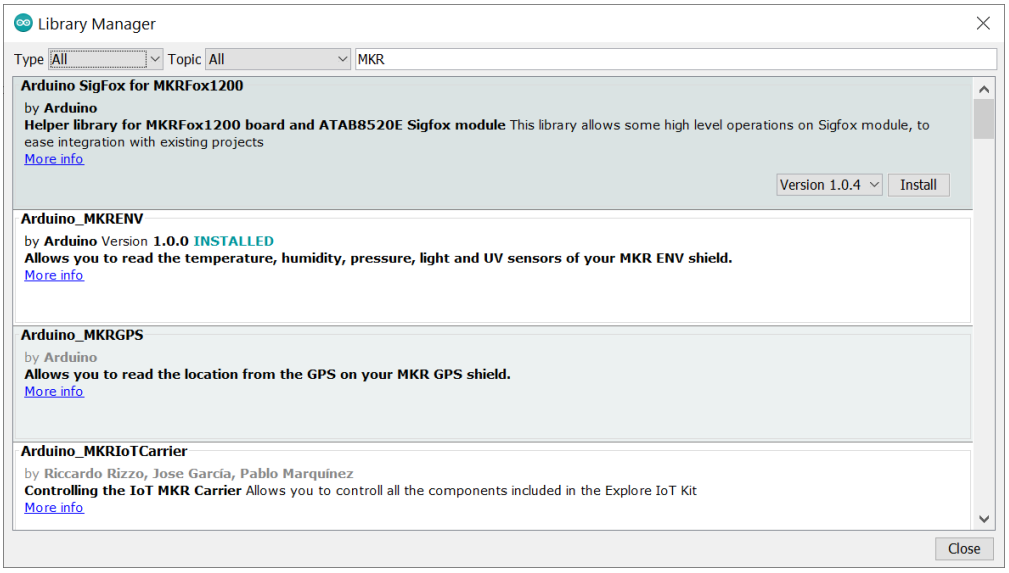
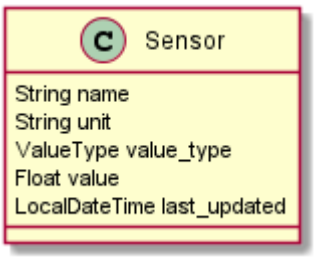


Figure 2. Arduino IDE Library Manager

Architecture

Sensor model



Sensors

Sensorlist

The list of available sensors is initialized using a JSON file. In this file all sensors and their required attributes are described. So that the application is able to read the file correctly it must correspond exactly to the specified format.

The file contains an attribute called `sensors`. This is a list type and must contain all sensor definitions. A sensor definition is enclosed by a pair of opening and closing braces `{ ⋯ }`. Each definition requires a `name`, a `value_type` and a `unit`. The ordering of these is important. As value type any of the defined constants in the `ValueType` enum could be used.

Listing 1. sensorlist-example

```
{
  "sensors": [ ①
    { ②
      "name": "Sensor 1",
      "value_type": "FLOAT",
      "unit": "°C"
    },
    {
      "name": "Sensor 2",
      "value_type": "FLOAT",
      "unit": "°C"
    }
  ]
}
```

① Start of the sensor list

② Start of a sensor definition

Line breaks and spaces are reduntant. All attributes of a sensor could also be in one line. But for clearnesses it is recommended to use the same formating.

Problems

Confirmation dialog onClose

There are no good resources online for the implementation of a confirmation dialog on a close request. Due to this it took some time to implement this.

The main goal of the dialog is to ask the user is sure if he want to close the application. For example, if the application is still connected to an Arduino the user will be asked if the Arduino should be disconnected and the application closed.

The following code demonstrates a simple example how a confirmation dialog could be shown when the user clicks on the 'close' button. This is only a general usage example how the functionality could be implemented and not a real code sample from the application.

To show a dialog the `setOnCloseRequest` must be overridden. This could be done in the `onDock` function of the view.

Listing 2. Disable onCloseRequest

```
// ...

val close: Boolean = false

override fun onDock() {
  currentStage?.setOnCloseRequest { evt ->
    if (!close) evt.consume()
  }
}

// ...
```

The above example shows how close event could be disabled. If the event is `consumed` the application will not close. The next step is to replace the boolean variable by an alert dialog.

```
override fun onDock() {
  currentStage?.setOnCloseRequest { evt ->
    val alert = Alert(AlertType.CONFIRMATION)
    alert.title = "Close the application"
    alert.headerText = "Are you sure you want to close the application?"
    alert.contentText = "You have some unsaved stuff. Are you sure you want to continue?"

    val okButton = ButtonType("Yes", ButtonBar.ButtonData.YES)
    val noButton = ButtonType("No", ButtonBar.ButtonData.NO)

    alert.buttonTypes.setAll(okButton, noButton)

    val result = alert.showAndWait()
    if (result.get() == okButton) {
      // ...
    } else {
      evt.consume()
    }
  }
}
```

Build the documentation

Use: `asciidoctor-pdf -r asciidoctor-diagram documentation.adoc`

Install rouge

```
gem install rouge
```