# **Microfire LLC Mod-EC Arduino Library**

## 

## **Release Information**

Copyright © 2023 Microfire LLC

This documentation is licensed under a [Creative Commons Attribution-NoDerivatives 4.0 International (CC BY-ND)](https://creativecommons.org/licenses/by-nd/4.0/).

## **Release History**

| **Release** | **Date** | **Description** |
| --- | --- | --- |
| 2.0.0 | 3/24/2023 | Changes for hardware version 2 |
| 1.0.0 | 4/23/2021 | Initial |

## **Legal Disclaimer**

TECHNICAL AND RELIABILITY DATA FOR MICROFIRE LLC PRODUCTS (INCLUDING DATASHEETS) AS MODIFIED FROM TIME TO TIME (“RESOURCES”) ARE PROVIDED BY MICROFIRE LLC "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW IN NO EVENT SHALL MICROFIRE LLC BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THE RESOURCES, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

MICROFIRE LLC reserves the right to make any enhancements, improvements, corrections or any other modifications to the RESOURCES or any products described in them at any time and without further notice.

The RESOURCES are intended for skilled users with suitable levels of design knowledge. Users are solely responsible for their selection and use of the RESOURCES and any application of the products described in them. User agrees to indemnify and hold MICROFIRE LLC harmless against all liabilities, costs, damages or other losses arising out of their use of the RESOURCES.

HIGH RISK ACTIVITIES. MICROFIRE LLC products are not designed, manufactured or intended for use in hazardous environments requiring fail safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, weapons systems or safety-critical applications (including life support systems and other medical devices), in which the failure of the products could lead directly to death, personal injury or severe physical or environmental damage, or business loss (“High Risk Activities”). MICROFIRE LLC specifically disclaims any express or implied warranty of fitness for High Risk Activities and accepts no liability for use or inclusions of MICROFIRE LLC products in High Risk Activities.

[Microfire LLC Mod-EC Arduino Library](#_xbpmu8527wlw)

[Release Information](#_9ell9us2g3yu)

[Release History](#_7mz16pu1fjvk)

[Legal Disclaimer](#_ts75w4sg8hm2)

[Library Documentation](#_6gb3jj3vx8ys)

[Installation](#_z1geqptyk2ky)

[Member Variables](#_hphrbgkppl11)

[const float S](#_6bb05gu961sc)

[const float mS](#_py9p4mp3doac)

[const float uS](#_p0bkoxd5jdfd)

[const float PSU](#_d9frf3dd99c5)

[const float density](#_vowtsx69xc26)

[const int PPM\_500](#_32lfj92dewgf)

[const int PPM\_640](#_ybdd5654a8fb)

[const int PPM\_700](#_nclow1o2n6bn)

[const float calibrationLowReading](#_gbgu1rvzvfs6)

[const float calibrationLowReference](#_11to662k4e1w)

[const float calibrationMidReading](#_uiim96mptv90)

[const float calibrationMidReference](#_yfaw0krtsf2v)

[const float calibrationHighReading](#_y13r0t3077dl)

[const float calibrationHighReference](#_fqubx53yam2x)

[const float calibrationSingleOffset](#_z7izshvyhlto)

[const int hwVersion](#_cla5b1yopezo)

[const int fwVersion](#_86cudup0mjj0)

[const int status](#_tni6b2b8j9r)

[Member Methods](#_cpct6d4k9akh)

[begin](#_xmaezoebo0jf)

[Definition](#_sa2r8ngpjaxx)

[Parameters](#_goe7pqmbfad2)

[Return](#_78phd5kjos9k)

[Example 1](#_f1lv9ueanh8s)

[Example 2](#_um52e9btjb19)

[connected](#_n71yxxdvyqai)

[Definition](#_7ma21ro0h7jt)

[Parameters](#_dhbnalrk81y4)

[Return](#_rhzbvwttb3tv)

[Example](#_u7fju1rtmk63)

[calibrateLow](#_5sc3d3e5bfbu)

[Definition](#_3yc8eswmg6kd)

[Parameters](#_1kub48hpwhu9)

[Return](#_99ck5e39q1uu)

[Example 1](#_2vh27p515ejo)

[Example 2](#_heme3qb2w39c)

[calibrateMid](#_o9zu32edqnhp)

[Definition](#_o3p10jp7642k)

[Parameters](#_ssh43gec9jm0)

[Return](#_8s088sk4xi6s)

[Example](#_t7rpef5xyr5g)

[calibrateHigh](#_6lbk55wkgk1j)

[Definition](#_8vmu7xnpfq6x)

[Parameters](#_o3rdornp0k6e)

[Return](#_yubykvzcygx4)

[Example](#_z2eoy257v2oq)

[calibrateSingle](#_gy0ogy8zudjv)

[Definition](#_f1icfz23jj7c)

[Parameters](#_dpl6orik6hrc)

[Return](#_4ik9j5nodlon)

[Example](#_y1wef2fsn9kz)

[getDeviceInfo](#_w2wqsmwki14y)

[Definition](#_v19bqiigthh8)

[Parameters](#_2bvvy2yyrv44)

[Return](#_cjbsli9e68cw)

[Example](#_r8mbjhmggj7f)

[measureEC](#_5lgktot395dt)

[Definition](#_jcle7qup79hd)

[Parameters](#_vhdll0b046sc)

[Return](#_uzywxulmpuzl)

[Example](#_2qip2fx9cksu)

[reset](#_7tt8oqmgdtjv)

[Definition](#_lwry98u2pucb)

[Parameters](#_bzh5bnu2zp2h)

[Return](#_deqojykhgno6)

[Example](#_hbaqpb330tfs)

[setDeviceInfo](#_iln1iekvzs1x)

[Definition](#_saear0zdjqot)

[Parameters](#_d0jxa94gjj71)

[Return](#_x7qt73sg6nps)

[Example](#_sxe0uz5xlhsp)

[setI2CAddress](#_hqkhov3noox4)

[Definition](#_pbanh548vhls)

[Parameters](#_vb4ne1576jan)

[Return](#_6m69cbbb7640)

[Example](#_1anzvbrt5yed)

[update](#_k98nd52qmd7r)

[Definition](#_9w1vns5wirj0)

[Parameters](#_sv3uwmnwg0u)

[Return](#_phlp9t890wcf)

[Example](#_v91qvhift0b4)

# Library Documentation

## Installation

Installation of this library can be done from within the Arduino IDE’s library manager. Search for `Microfire` and choose the library labeled `Microfire Mod-EC`.

The library is also installable through PlatformIO’s library manager.

## Member Variables

### const float S

Result of the last EC measurement in Siemens.

### const float mS

Result of the last EC measurement in millisiemens.

### const float uS

Result of the last EC measurement in microsiemens.

### const float PSU

Result of the last EC measurement in Practical Salinity Units.

### const float density

The density of saline water in g/cm3.

### const int PPM\_500

Result of the last EC measurement in TDS, calculated using a coefficient of 500.

### const int PPM\_640

Result of the last EC measurement in TDS, calculated using a coefficient of 640.

### const int PPM\_700

Result of the last EC measurement in TDS, calculated using a coefficient of 700.

### const float calibrationLowReading

Dual-point low reading of the probe. If there is no calibration data present, NaN (not a number) is returned.

### const float calibrationLowReference

Dual-point low reference of the probe. If there is no calibration data present, NaN (not a number) is returned.

### const float calibrationMidReading

Triple-point middle reading of the probe. If there is no calibration data present, NaN (not a number) is returned.

### const float calibrationMidReference

Triple-point middle reference of the probe. If there is no calibration data present, NaN (not a number) is returned.

### const float calibrationHighReading

Dual-point high reading of the probe. If there is no calibration data present, NaN (not a number) is returned.

### const float calibrationHighReference

Dual-point high reference of the probe. If there is no calibration data present, NaN (not a number) is returned.

### const float calibrationSingleOffset

Single offset calibration data. If there is no calibration data present, NaN (not a number) is returned.

### const int hwVersion

Hardware version of the module.

### const int fwVersion

Firmware version of the module.

### const int status

Status code of the last measurement or calibration.

**0**: STATUS\_NO\_ERROR

**1**: STATUS\_NO\_PROBE\_RANGE

**2**: STATUS\_SYSTEM\_ERROR

**3**: STATUS\_CONFIG\_ERROR

## Member Methods

### **begin**

Initializes the library and determines if the module is connected. Wire.begin() must be called prior.

#### Definition

| **bool begin(TwoWire &wirePort = Wire, uint8\_t address = 0x0A);** |
| --- |

#### Parameters

| **Parameter** | **Description** |
| --- | --- |
| **&wirePort** | TwoWire I2C interface |
| **address** | I2C address of the module |

#### Return

| **Type** | **Description** |
| --- | --- |
| **bool** | **True** if the module is connected.  **False** if the module is disconnected. |

#### Example 1

| **#include <Microire\_Mod-EC.h>** Microire::Mod\_EC::i2c ec;  **void** **setup**() {  Wire.begin();  **if** (ec.begin() != true)  {  *// Error: the sensor isn't connected*  } }  **void** **loop** () {} |
| --- |

#### Example 2

| **#include <Microfire\_Mod-EC.h>** Microfire::Mod\_EC::i2c ec;  **void** **setup**() {  Wire.begin();  **if** (ec.begin(Wire, 0xA0) != true)  {  *// Error: the sensor isn't connected*  } }  **void** **loop** () {} |
| --- |

### **connected**

Determines if the module is connected.

#### Definition

| **bool** **connected**(); |
| --- |

#### Parameters

| **Parameter** | **Description** |
| --- | --- |
| **None** |  |

#### Return

| **Type** | **Description** |
| --- | --- |
| **bool** | **True** if the module is connected.  **False** if the module is disconnected. |

#### Example

| **#include <Microfire\_Mod-EC.h>** Microfire::Mod\_EC::i2c ec;  **void** **setup**() {  Wire.begin();  ec.begin();  **if** (ec.connected() != true)  {  *// Error: the sensor isn't connected*  } }  **void** **loop** () {} |
| --- |

### 

### **calibrateLow**

Performs a low-point calibration. [Status](#_tni6b2b8j9r), [calibrationLowReference](#_11to662k4e1w), and [calibrationLowReading](#_gbgu1rvzvfs6) are updated. It takes 750 ms to complete a measurement.

#### Definition

| **float** **calibrateLow**(**float** solutionEC, **float** tempC = 25.0, **float** tempCoef = 0.019, **float** tempConst = 25.0, **float** k = 1.0, **bool** blocking = true); |
| --- |

#### Parameters

| **Parameter** | **Description** |
| --- | --- |
| **solutionEC** | The conductivity of the calibration solution in mS/cm² |
| **tempC** | The calibration solution’s temperature in Celsius |
| **tempCoef** | The temperature coefficient used for temperature compensation. Typically 0.019 for freshwater and 0.021 for seawater |
| **tempConst** | The temperature constant used for temperature compensation |
| **k** | The cell-constant, or K value of the attached EC probe. Typically 0.1, 1.0 or 10.0 |
| **blocking** | Return immediately or wait for the module to complete the calibration |

#### Return

| **Type** | **Description** |
| --- | --- |
| **uint8\_t** | An error code for the measurement. Can be one of the following:  **0**: no error  **1**: no probe detected or outside range  **2**: system error  **3**: config error |

#### Example 1

| **#include <Microfire\_Mod-EC.h>** Microfire::Mod\_EC::i2c ec;  **void** **setup**() {  Wire.begin();   ec.begin();  ec.calibrateLow(0.7, 23.1);  **if** (ec.status)  {  *// Error*  } }  **void** **loop** () {} |
| --- |

#### Example 2

| **#include <Microfire\_Mod-EC.h>** Microfire::Mod\_EC::i2c ec;  **void** **setup**() {  Wire.begin();   ec.begin();  **if** (ec.calibrateLow(0.7, 23.1))  {  *// Error*  } }  **void** **loop** () {} |
| --- |

### 

### **calibrateMid**

Performs a mid-point calibration. [Status](#_tni6b2b8j9r), [calibrationMidReference](#_yfaw0krtsf2v), and [calibrationMidReading](#_uiim96mptv90) are updated. It takes 750 ms to complete a measurement.

#### Definition

| **float** **calibrateMid**(**float** solutionEC, **float** tempC = 25.0, **float** tempCoef = 0.019, **float** tempConst = 25.0, **float** k = 1.0, **bool** blocking = true); |
| --- |

#### Parameters

| **Parameter** | **Description** |
| --- | --- |
| **solutionEC** | The conductivity of the calibration solution in mS/cm² |
| **tempC** | The calibration solution’s temperature in Celsius |
| **tempCoef** | The temperature coefficient used for temperature compensation. Typically 0.019 for freshwater and 0.021 for seawater |
| **tempConst** | The temperature constant used for temperature compensation |
| **k** | The cell-constant, or K value of the attached EC probe. Typically 0.1, 1.0 or 10.0 |
| **blocking** | Return immediately or wait for the module to complete the calibration |

#### Return

| **Type** | **Description** |
| --- | --- |
| **uint8\_t** | An error code for the measurement. Can be one of the following:  **0**: no error  **1**: no probe detected or outside range  **2**: system error  **3**: config error |

#### Example

| **#include <Microfire\_Mod-EC.h>** Microfire::Mod\_EC::i2c ec;  **void** **setup**() {  Wire.begin();   ec.begin();  ec.calibrateMid(1.413, 23.1);  **if** (ec.status)  {  *// Error*  } }  **void** **loop** () {} |
| --- |

### 

### **calibrateHigh**

Performs a high-point calibration. [Status](#_tni6b2b8j9r), [calibrationHighReference](#_fqubx53yam2x), and [calibrationHighReading](#_y13r0t3077dl) are updated. It takes 750 ms to complete a measurement.

#### Definition

| **float** **calibrateHigh**(**float** solutionEC, **float** tempC = 25.0, **float** tempCoef = 0.019, **float** tempConst = 25.0, **float** k = 1.0, **bool** blocking = true); |
| --- |

#### Parameters

| **Parameter** | **Description** |
| --- | --- |
| **solutionEC** | The conductivity of the calibration solution in mS/cm² |
| **tempC** | The calibration solution’s temperature in Celsius |
| **tempCoef** | The temperature coefficient used for temperature compensation. Typically 0.019 for freshwater and 0.021 for seawater |
| **tempConst** | The temperature constant used for temperature compensation |
| **k** | The cell-constant, or K value of the attached EC probe. Typically 0.1, 1.0 or 10.0 |
| **blocking** | Return immediately or wait for the module to complete the calibration |

#### Return

| **Type** | **Description** |
| --- | --- |
| **uint8\_t** | An error code for the measurement. Can be one of the following:  **0**: no error  **1**: no probe detected or outside range  **2**: system error  **3**: config error |

#### Example

| **#include <Microfire\_Mod-EC.h>** Microfire::Mod\_EC::i2c ec;  **void** **setup**() {  Wire.begin();   ec.begin();  ec.calibrateHigh(10.0, 23.1);  **if** (ec.status)  {  *// Error*  } }  **void** **loop** () {} |
| --- |

### 

### **calibrateSingle**

Performs a single-point calibration. [status](#_tni6b2b8j9r) and [calibrationSingleOffset](#_z7izshvyhlto) are updated. It takes 750 ms to complete a measurement.

#### Definition

| **float** **calibrateSingle**(**float** solutionEC, **float** tempC = 25.0, **float** tempCoef = 0.019, **float** tempConst = 25.0, **float** k = 1.0, **bool** blocking = true); |
| --- |

#### Parameters

| **Parameter** | **Description** |
| --- | --- |
| **solutionEC** | The conductivity of the calibration solution in mS/cm² |
| **tempC** | The calibration solution’s temperature in Celsius |
| **tempCoef** | The temperature coefficient used for temperature compensation. Typically 0.019 for freshwater and 0.021 for seawater |
| **tempConst** | The temperature constant used for temperature compensation |
| **k** | The cell-constant, or K value of the attached EC probe. Typically 0.1, 1.0 or 10.0 |
| **blocking** | Return immediately or wait for the module to complete the calibration |

#### Return

| **Type** | **Description** |
| --- | --- |
| **uint8\_t** | An error code for the measurement. Can be one of the following:  **0**: no error  **1**: no probe detected or outside range  **2**: system error  **3**: config error |

#### Example

| **#include <Microfire\_Mod-EC.h>** Microfire::Mod\_EC::i2c ec;  **void** **setup**() {  Wire.begin();   ec.begin();  **if** (ec.calibrateSingle(1.413, 23.1) != ec.STATUS\_NO\_ERROR)  {  *// Error*  } }  **void** **loop** () {} |
| --- |

### 

### **getDeviceInfo**

Updates all measurement, calibration, and system registers with the most recent information.

#### Definition

| **void** **getDeviceInfo**(); |
| --- |

#### Parameters

| **Parameter** | **Description** |
| --- | --- |
| **None** |  |

#### Return

| **Type** | **Description** |
| --- | --- |
| **None** |  |

#### Example

| **#include <Microfire\_Mod-EC.h>** Microfire::Mod\_EC::i2c ec;  **void** **setup**() {  Wire.begin();   ec.begin();  ec.getDeviceInfo();  *// The following variables are updated:*  *// ec.hwVersion, ec.fwVersion*  *// ec.calibrationLowReference, ec.calibrationLowReading*  *// ec.calibrationMidReference, ec.calibrationMidReading*  *// ec.calibrationHighReference, ec.calibrationHighReading*  *// ec.calibrationSingleOffset* }  **void** **loop** () {} |
| --- |

### 

### **measureEC**

Starts an EC measurement. This function provides default values for measuring freshwater (temperature coefficient of 0.019, probe cell-constant of 1.0).

Member variables [S](#_6bb05gu961sc), [mS](#_py9p4mp3doac), [uS](#_p0bkoxd5jdfd), [PSU](#_d9frf3dd99c5), [density](#_vowtsx69xc26), [PPM\_500](#_32lfj92dewgf), [PPM\_640](#_ybdd5654a8fb), [PPM\_700](#_nclow1o2n6bn), and [status](#_tni6b2b8j9r) are updated. [tempC](#_r1dhx9kooeu2) is modified to either what was passed or to the default of 25.0.

#### Definition

| **float** **measureEC**(**float** tempC = 25.0, **float** tempCoef = 0.019, **float** tempConst = 25.0, **float** k = 1.0, **float** kPa = 0, **bool** blocking = true); |
| --- |

#### Parameters

| **Parameter** | **Description** |
| --- | --- |
| **tempC** | The solution-under-test’s temperature in Celsius. |
| **tempCoef** | The temperature coefficient used for temperature compensation. Typically 0.019 for freshwater and 0.021 for seawater. |
| **tempConst** | The temperature constant used for temperature compensation |
| **k** | The cell-constant, or K value of the attached EC probe. Typically 0.1, 1.0 or 10.0 |
| **kPa** | The pressure in kilopascals at which the measurement is being made. Used in salinity and density calculations. If salinity or density measurements aren’t needed, 0 should be used. |
| **blocking** | Return immediately or wait for the module to complete the measurement |

#### Return

| **Type** | **Description** |
| --- | --- |
| **float** | The solution-under-test’s conductivity in mS/cm. |

#### Example

| **#include <Microfire\_Mod-EC.h>** Microfire::Mod\_EC::i2c ec;  **void** **setup**() {  Wire.begin();   ec.begin();  ec.measureEC(23.1);  *// Measurement results are in*  *// ec.S, ec.mS, ec.uS, ec.PSU, ec.density,*   *// ec.PPM\_500, ec.PPM\_640, ec.PPM\_700* }  **void** **loop** () {} |
| --- |

### 

### **reset**

Resets all calibration data to the empty value of NaN (not a number).

#### Definition

| **void** **reset**(); |
| --- |

#### Parameters

| **Parameter** | **Description** |
| --- | --- |
| **None** |  |

#### Return

| **Type** | **Description** |
| --- | --- |
| **None** |  |

#### Example

| **#include <Microfire\_Mod-EC.h>** Microfire::Mod\_EC::i2c ec;  **void** **setup**() {  Wire.begin();   ec.begin();  ec.reset(); }  **void** **loop** () {} |
| --- |

### 

### **setDeviceInfo**

Sets all the device calibration registers with a specified value.

#### Definition

| **void** **setDeviceInfo**(**float** calibrationLowReading, **float** calibrationLowReference, **float** calibrationMidReading, **float** calibrationMidReference, **float** calibrationHighReading, **float** calibrationHighReference, **float** calibrationSingleOffset); |
| --- |

#### Parameters

| **Parameter** | **Description** |
| --- | --- |
| **calibrationLowReading** | Read-low calibration data |
| **calibrationLowReference** | Reference-low calibration data |
| **calibrationMidReading** | Read-mid calibration data |
| **calibrationMidReference** | Reference-high calibration data |
| **calibrationHighReading** | Read-high calibration data |
| **calibrationHighReference** | Reference-high calibration data |
| **calibrationSingleOffset** | Single-offset calibration data |

#### Return

| **Type** | **Description** |
| --- | --- |
| **None** |  |

#### Example

| **#include <Microfire\_Mod-EC.h>** Microfire::Mod\_EC::i2c ec;  **void** **setup**() {  Wire.begin();   ec.begin();  ec.setDeviceInfo(8400.1, 10000.0, 9891.8, 1000.0, 99.1, 100.0, NAN); }  **void** **loop** () {} |
| --- |

### 

### **setI2CAddress**

Changes the I2C address of the module. The change is stored and used again after a power-cycle.

Note: The library will use the new I2C address after calling this method, but the address must be stored and [begin](#_xmaezoebo0jf) must be called with the new address on subsequent initialization.

#### Definition

| **void** **setI2CAddress**(**uint8\_t** i2cAddress); |
| --- |

#### Parameters

| **Parameter** | **Description** |
| --- | --- |
| **uint8\_t** | New I2C address |

#### Return

| **Type** | **Description** |
| --- | --- |
| **None** |  |

#### Example

| **#include <Microfire\_Mod-EC.h>** Microfire::Mod\_EC::i2c ec;  **void** **setup**() {  Wire.begin();   ec.begin();  ec.setI2CAddress(0x20); }  **void** **loop** () {} |
| --- |

### 

### **update**

If blocking is set to false when [measureEC](#_5lgktot395dt) is called, this method will update [S](#_6bb05gu961sc), [mS](#_py9p4mp3doac), [uS](#_p0bkoxd5jdfd), [PSU](#_d9frf3dd99c5), [density](#_vowtsx69xc26), [PPM\_500](#_32lfj92dewgf), [PPM\_640](#_ybdd5654a8fb), and [PPM\_700](#_nclow1o2n6bn). This allows the controlling device to do other work rather than wait for the module to complete the measurement.

#### Definition

| **void** **update**(); |
| --- |

#### Parameters

| **Parameter** | **Description** |
| --- | --- |
| **None** |  |

#### Return

| **Type** | **Description** |
| --- | --- |
| **None** |  |

#### Example

| **#include <Microfire\_Mod-EC.h>** Microfire::Mod\_EC::i2c ec;  **void** **setup**() {  Wire.begin();   ec.begin();  ec.measureEC(23.1, 0.019, 25.0, 1.0, 0.0, false);  *// blocking = false above, do other work for 750 ms*  ec.update();  *// Measurement results are in*  *// ec.S, ec.mS, ec.uS, ec.PSU, ec.density,*   *// ec.PPM\_500, ec.PPM\_640, ec.PPM\_700* }  **void** **loop** () {} |
| --- |

## 