# Application note: baby oak tree watering project.

1	Intr	oduc	tion	.2
	1.1	Obj	ective of this document	.2
	1.2	Gre	enhouse watering application functional description	.2
	1.3	Soft	ware and hardware requirements	.3
	1.4	Req	uired user experience	.4
2	"W	awiW	/aterSensorValve" live demo	.5
	2.1	Har	dware connections	.5
	2.2	Loa	d the demo	.5
	2.3	Visu	alize the variables of interest	.6
	2.4	Rea	ding and scaling the sensor input	.6
	2.5	Acti	vate moisture control (turn on watering)	.8
	2.6	Sim	ulate drought	.9
	2.7	Alar	m generation	11
	2.8	Way	wiLib as HMI	11
3	"W	awiW	/aterSensorValve" data recording demo	12
	3.1	Con	tinuous recording of the moisture level	12
	3.2	Cha	nge based recording of the state machine status.	16
4	"W	awiW	/aterSensorValve" Arduino software automation concepts	20
	4.1	Nev	er put the main loop on hold or in delay	20
	4.2	Way	wiTimer object	21
	4.2	.1	The concept "millis() without delay in a C++ object"	21
	4.2	.2	Short look under the hood	21
	4.3	The	finite state machine concept	23
	4.3	.1	Introduction	23
	4.3	.2	Finite state machine concept	23
	4.3	.3	Finite state machine implementation	<u>2</u> 4
	4.3	.4	Output control using finite state machines	26
	4.3	.5	Other parts	26
5	FUF	RTHEF	READING	26
6	API	PEND	IX: CODE LISTING	27

# 1 Introduction

# 1.1 Objective of this document

The objective of this document is to describe how to use WawiLib in a real automation example. The example is a program to control water distribution to a small oak tree based on the moisture level of the soil.

Each year, new oaks grow in my garden as the level of the soil has been raised with ground from a foreign location by the previous owner. The objective of this project is to make sure at least of one of them survives (normally they dry out during summer).

The application is built as a real industrial automation application. It contains reading and scaling of an analog input value, a finite state machine (aka "Grafcet") with automation logic and alarming. All parameters and controls can be modified using WawiLib.

The WawiLib "Getting started" demos use simple C language commands. This application takes us to another level: a framework for full-featured industrial automation applications. It shows how to implement multiple timers without blocking the main Arduino loop (not using delay()) and a safe and reliable way to implement sequential logic (finite state machine).

The use of breakpoints in the sketch to debug the application will be documented in this project note.

WawiLib will be used as a tool to test the application and as a tool to observe and operate the Arduino application. The demo uses USB for convenience but WiFi and Ethernet are also possible.

# 1.2 Greenhouse watering application functional description

The application uses a soil moisture sensor to read the moisture level of the soil. The moisture level is scaled into a range 0...100% into the variable *moistPctIst*. The scaling range can be modified using WawiLib to change *moistAnaMax* and *moistAnaMin. moistAna* contains the raw input value read directly from the analog input of the Arduino.



Fig 1.1. Baby oak tree in garden with moisture sensor.

Watering can be enabled or disabled setting the variable *wateringEnable to 1*. If watering is enabled, *moistPctSoll* is used as setpoint for the moisture level. If the soil gets too dry, a cycle will start where the water valve will be opened and closed repeatedly.

Watering has to be done carefully. Therefore, even if the soil is too dry, the water will only be opened for a limited period of time. After that the water valve will be closed to give the water time to settle. If the moisture level is still not OK, the cycle will restart. The watering cycle can be controlled using the variables *valveTimeOpenMSec* and *valveTimeHoldMSec*. The both of them contain time values in milliseconds.

If watering is enabled and the level of moisture does not reach the requested level after a certain period of time, an alarm output will signal the abnormal situation. The alarm state can be checked using the variable *dryAlarm*. The maximum time soil moisture can go below its setpoint before generating an alarm is determined by *moistureTimeAlarmSec*.

TotalValveTime contains the total time the water valve was open.

The control logic is implemented using a finite state machine. The state of the machine can be checked reading the value of *stateString*. The application contains various software timers. WawiLib can be used to read the actual values of these timers using the variables *timerFsm* and *dryAlarmTimer*. A detailed description of how the logic is implemented can be found later in this document.

State changes of the FSM are reported in the output window so the user/developer can follow what is going on in a chronological order.

## 1.3 Software and hardware requirements

The Arduino IDE (in this example 1.8.15) and WawiLib both need to be installed on your PC. The demo runs with the licensed versions of WawiLib. With the unlicensed version you can use the demo concepts with 1 variable at a time.

Essential hardware you need is an Arduino (Mega) board, a USB programming cable, some Dupont male-male (breadboard) wires, a glass of water and a Windows PC (32 or 64 bit).

You also need a sensor that generates an analog signal 0-5V to determine the moisture level of the soil. In this example I use a sensor as in the fig. 1.2. It is an LM393 based converter. This type of sensor and conversion devices are available at many suppliers on the Internet. These signal amplifiers use an operational amplifier to create a voltage signal based on the resistance of the soil.



Fig. 1.2. Various types of moisture sensors.

(If the hardware is not available you can use a potentiometer instead to simulate the output voltage of the sensor.)

The program uses I/O13 to control the valve (or a pump) and I/O12 to issue an alarm signal if moisture control is failing (soil is too dry for too long.)

The last thing you need is a glass of water to dip the sensor in to check if your application is working properly.

In this demo we will use the Arduino MEGA2560 board but other boards can be used in a similar or even identical way. (Be careful: some Arduino's are 3.3V and others are 5V based)

# 1.4 Required user experience

You should be familiar with the tutorials "Getting started with WawiLib USB" and "Debugging with WawiLib USB". There are no specific additional requirements. The C code used in this example requires knowledge of the C language that is a bit more extended compared to the "Getting stared" examples.

The reason is that the concepts presented in this application can be used as a framework for other automation applications. I will first give a demo of the application as a whole and in the following chapters I will describe in detail the different parts of the application.

# 2 "WawiWaterSensorValve" live demo

## 2.1 Hardware connections

- ✓ Connect the GND pin of the interface converter to the GND of the Arduino (Pins next to pin 53).
- $\checkmark$  Connect the VCC pin of the interface converter to the +5V of the Arduino (Pins next to pin 22).
- ✓ Connect the moisture sensor to the 2 sensing pins on the interface converter (one of them has an "earth" symbol next to them).
- ✓ Connect the AO output signal of the interface converter to A8 of the Arduino (analog input 8).
- ✓ Connect the Arduino Mega to the PC using the USB cable.



Fig. 2.1. Arduino Mega + sensor connections.

# 2.2 Load the demo

- ✓ Download the WawiWaterSensorValve.ino application from www.SylvesterSolutions.com. You can find the application in the download section of <u>www.sylvestersolutions.com.</u>
- ✓ Compile the demo and upload it to the Arduino board.

# 2.3 Visualize the variables of interest

## ✓ Start WawiLib and monitor the variables as indicated in the table below.

🔊 🖻 New Open	Save	👼 Print	Copy	K Cut	Deste Paste	offline	Setup()	Loop()	Write all	▶ Continue	• brkpt	Ø brkp	ot	
vailable variables				Interface/	Ard. ID	Variabl	e name		Actual v	alue	Write	value	Format	
ser1\MyArduino		-	1	ser1/MvA	rduino	waterin	aEnable		060000'0	000	0		BIT	@wateringE
+ moistAna			2				5							- 5
+ moistAnaMin			3	ser1/MyA	rduino	mois	tAna		1023				INT	@moistAna
+ moistAnaMax			4	ser1/MyA	rduino	moistA	naMin		300				INT	@moistAna
+ moistPctSoll			5	ser1/MyA	rduino	moistA	naMax		1024				INT	@moistAna
. ■ moistPctIst			6											
wateringEnable	e		7	ser1/MyA	rduino	moist	PctIst		0.1381	23			FLOAT	@moistPctl
+ valveOpen			8	ser1/MyA	rduino	moist	PctSoll		50				FLOAT	@moistPct
• dryAlarm			9											
timerFsm ±			10	ser1/MyA	rduino	stateStri	ng[029]	STATE_	0_VALVE_	WATER_OFF			STRING	@stateStrin
+ dryAlarmTime	r		11											
+ stateCurrent	N/C		12	ser1/MyA	rduino	dryAlar	mTimer		0				INT	@dryAlarm
valve TimeOpe	enivisec		13											
	MSoc		14	ser1/MyA	rduino	valveTime	OpenMSec		10000	)	1000	00	INT	@valveTim
totalValveTim	ar		15	ser1/MyA	rduino	valveTime	HoldMSec		60000	)	1000	00	INT	@valveTim
Array			16											
+ stateString[0	291		17	ser1/MyA	rduino	time	rFsm		0				INT	@timerFsm
ji	,		18											
			19	ser1/MyA	rduino	valve	Open		0					@valveOpe
			20											
		<												3
. Time			Node			Mess	sage							
3/08/2021 14:54	4:15.487		ser1/CON	//33/MyArd	uino	Closi	ng commu	nications	port					
1 3/08/2021 14:5	5:27.340		ser1/CON	//33/MyArd	uino	Oper	ning serial p	ort [CON	133] Ok					
2 3/08/2021 14:5	5:27.340		ser1/CON	/33/MyArd	uino	Setti	ng port par	ameters [	baud=115	200 parity=1	data=8	stop=	1] OK	
3/08/2021 14:5	5:28.363		ser1/CON	/33/MyArd	uino	Read	ling settings	from Are	duino:	•				
4 3/08/2021 14.5	5:28.377		ser1/COM	133/MyArd	uino	Ardu	ino library	version 02	2.01. Ardu	no buffer siz	e: (RX.T)	() = (64)	64) bytes	Communi

Fig. 2.2. Sketch variabeles of interest.

# 2.4 Reading and scaling the sensor input

✓ Put the sensor in a glass of water

BV	VawiLib-PC [C:\I	Users\Jc	ohi∖	Docum	ents\No	Name.Wif*]-	[C:\Users\.	Johi\Docun	nents\Well	Data\Well	1.bak.Wvf	*]	[	<u>_</u>		×	٦
File	Edit Settings H	lelp															
<u>@</u>	Ð			8	0	X	۵	1	6	6	4	Þ	٠	Ø			
Ne	w Open	Save		Print	Сору	Cut	Paste	Offline	Setup()	Loop()	Write all	Continue	brkpt	brkp	ot		
Ava	ailable variables		^			Interface/	Ard. ID	Variabl	e name		Actual v	alue	Write	value	Format		^
	Single variable				2												
T	+ moistAna				3	ser1/MyA	rduino	mois	stAna		259				INT	@mois	
	+ moistAnaMin	( <b>-</b>			4	ser1/MyA	rduino	moistA	AnaMin		300		23	81	INT	@mois	
	+ moistAnaMax	<			5	ser1/MyA	rduino	moistA	naMax		1024	1			INT	@mois	
	+ moistPctSoll				6												
	+ moistPctlst				7	ser1/MyA	rduino	moist	tPctlst		105.66	53			FLOAT	@moi:	
	wateringEnab	ole			8	ser1/MyA	rduino	moist	PctSoll		50				FLOAT	@mois	
			~		٥						_						~
<		>		<												>	
In	Time				Node			Mes	sage								^
000	3/08/2021 14:5	54:15.48	7		ser1/COM	//33/MyArdu	uino	Clos	ing comm	unications	port						
001	3/08/2021 14:5	5:27.34	0		ser1/COM	/////////////	uino	Ope	ning serial	port [CON	133] Ok	<					
002	3/08/2021 14:5	5:27.34	0		ser1/COM	/////////////	uino	Setti	ng port pa	rameters	[baud=115	5200 parity=	N data=	8 stop=	=1] OK		-
<																>	
		Loop	0	Autow	rite on	No recorde	ers active	ser1=My	Arduino=C	OM33/11	5200,8,N,1	,AVR [ITF_LC	OOP] msg	.ok/tot	: 107390	/107405	

Fig. 2.3. moistAna changes when sensor is in glass of water.

- You should see the value of *moistAna* go down (= analog input 0..1023). Low resistance (= water between the electrodes) makes the value go down until 259 in my case (fig 2.3). The output value of the signal converter is about 1.05V (Fluke multimeter measurement). As you go deeper with the sensor in the water, the value decreases.
- ⇒ The scaled value *moistPctIst* (moisture percentage calculated) will vary as the value of *moistAna* changes. But *moistPctIst* goes up as *moistAna* goes down. So *moistPctIst* increases as the soil becomes wetter. The idea is that moistPctIst is 100% for completely wet soil.

New	©pen	<b>■</b> Save		Print Co	ру	Cut	Paste	Offline	Setup()	Loop()	Write all	► Continue	• brkpt	⊠ brkp	ot	
Availat	ole variables		^			Interface/	Ard. ID	Variabl	e name		Actual v	alue	Write	value	Format	
= ser1	MyArduino			2												
= Sin				3		ser1/MyA	rduino	mois	tAna		1023	3			INT	@mois
1 m	noistAnaMin			4		ser1/MyA	rduino	moistA	AnaMin		300		23	1	INT	@mois
+ m	noistAnaMax			5		ser1/MyA	rduino	moistA	naMax		1024	4			INT	@mois
Ξn	noistPctSoll			6												
± m	noistPctIst			7		ser1/MyA	rduino	moist	Pctlst		0.1381	23			FLOAT	@moi
ŧw	ateringEnab	le		8		ser1/MyA	rduino	moist	PctSoll		50				FLOAT	@moi
	-luconen	>	Ť	<												>
Tin	ne			Node				Mes	sage							
00 3/0	08/2021 14:5	4:15.48	7	ser1/0	ON	M33/MyArdu	uino	Closi	ing comm	unications	port					
01 3/0	8/2021 14:5	5:27.34	0	ser1/0	ON	M33/MyArdu	uino	Oper	ning serial	port [COI	M33] OI	K				
02 2/0	8/2021 14.5	5:27.34	0	ser1/0		M33/MvArdu	uino	Setti	ng port pa	rameters	[baud=11]	5200 parity=	N data=8	stop=	11 OK	

 $\checkmark$  Take the sensor out of the water (keep it in the air).

Fig. 2.4. moistAna when isolated (not in the water).

- ⇒ You should see the value of *moistAna* rise (= analog input 0..1023). High resistance (= no water) makes the value go up until 1023, the maximum value of the A/D on the Arduino board. In my case the output value of the signal converter is 4.96V (measured with my Fluke 117.)
- $\Rightarrow$  Look at the value of *moistPctIst* = the actual moisture value, it goes down to practically 0%.
- ✓ Put the sensor back in the glass of water.
- ✓ Vary the *moistAnaMin* and *moistAnaMax* scaling factors. (Use WawiLib to change the value to these parameters.)

P New	Dpen		🖶 Print	Сору	Cut	De Paste	Offline	Setup	0 Loo	0 Write all	
		Variable	e name	Act	tual value		Write val	ue	Format	Variable address and sta	tus
1		moist	Ana		387				INT	@moistAna=0x03DC [2 byte] x 1 VAR_READING_QK -	_
2		moistA	naMin		200		200		INT	@moistAnaMin=0x020A [2 byte] x 1 VAR_READIMS_OK	-
3		moistA	naMax		1024				INT	@moistAnaMax=0x020C [2 byte] x 1 VAR_READING_OK	-
4											
5		moist	octist	1	77.3058				float	@moistPctIst=0x03D8 [4 byte] x 1 VAR_READING_OK -	
6		moistP	ctSoll		50				float	@moistPctSoll=0x0206 [4 byte] x 1 VAR_READING_OK -	
:											>
Index	Time			Node			Message	2			
							10 01			· · · · · · · · · · · · · · · · · · ·	

Fig. 2.5. *moistAna* when the sensor is in the water.

⇒ If you change the scaling factors *moistAnaMin* and *moistAnaMax*, you will see the calculated percentage of moisture change in line with the new scaling factors.

#### 2.5 Activate moisture control (turn on watering)

- ✓ Look at the values of *wateringEnable* and *stateString*.
- ⇒ Watering is disabled and the FSM (finite state machine) is in the state "STATE\_0\_VALVE\_WATER\_OFF".

2	VawiLib-PC [C:\U	sers\Jc	ohi∖	Docum	ents\NoN	Name.Wif*]-	[C:\Users\	Johi\Docur	nents\Wel	Data∖Wel	1.bak.Wvf	*]		-		1 ×	<
File	Edit Settings He	elp															
2 No		Savo		Brint	Conv	ж Сut	Basto	Offling	Totup()	Toop()	₩rito all	Continuo	brknt	Ø	nt		
ive	w Open	Save		Print	Сору	Cut	Paste	Onnine	Setup()	L00p()	write all	Continue	ыкрі	DIK	ρι	1	-
	+ moistAna		^			Interface/	Ard. ID	Variable	e name		Actual val	ue	Write va	lue l	Format		Ê
	+ moistAnaMax				1	ser1/MyA	rduino	watering	Enable		060000,00	00	0		BIT	@water	ii
	+ moistPctSoll				2	ser1/MyA	rduino	moist	Ana		1023				INT	@moist	:/
	<b>moistPctIst</b>				3	ser1/MyA	rduino	moistA	naMin		300		231		INT	@moist	: <b>/</b>
	wateringEnable	е			4	ser1/MyA	rduino	moistA	naMax		1024				INT	@moist	4
	<mark>≇ valveOpen</mark>				5	ser1/MyA	rduino	moist	Pctlst		0.138123	3			FLOAT	@moist	F
	+ dryAlarm				6	ser1/MyA	rduino	moistP	ctSoll		50				FLOAT	@moist	:F
	+ timerFsm				7	ser1/MyA	rduino	stateCu	urrent		0					@stateC	2
	+ dryAlarm I imei	r			8	ser1/MyA	rduino	stateStrin	ng[029]	STATE_C	_VALVE_W	ATER_OFF		5	STRING	@stateS	51
<	stateCurrent	>	Ť	<	9						-					>	Ý
					NI 1												
In	lime		_		Node			Mes	sage								^
000	3/08/2021 14:54	1:15.48	7		ser1/CON	M33/MyArdu	uino	Clos	ing comm	unications	port						
001	3/08/2021 14:55	5:27.34	0		ser1/CON	M33/MyArdu	uino	Ope	ning serial	port [COI	M33] Ok	<					~
<																>	
		Loop	0	Autow	rite on	No recorde	ers active	ser1=My	Arduino=C	OM33/11	5200,8,N,1	,AVR [ITF_L	OOP] msg.	.ok/to	t: 10933	37/10935	2

Fig. 2.6. Watering wateringEnable turned off.

- ✓ Put the sensor in the water
- ✓ Make sure *moistPctIst* is above *moistPctSolI* (simulate soil wetter than necessary)
- ✓ Enable "display .print() messages" in the WawiLib output window.
- ✓ Write the value 1 to the variable *wateringEnable* via WawiLib.

Mour	Onon	E .		Brint	Conv	ж Сит	Bacto	Offline	To Sotup()	Toop()	₩/rito all	Continuo	brknt	h	⊠ kat		
New	moistAna	Save	^	Print	Сору	Interface/	Ard. ID	Variable	e name	LOOP()	Actual valu	Je	Write v	alue	Format		^
÷.	moistAnaiviin moistAnaMax		-		1	ser1/MyA	rduino	watering	JEnable	1	0Ь0000'000	01	1		BIT	@water	ii
	moistPctSoll				2	ser1/MyA	rduino	moist	Ana		271				INT	@moist	:/
Đ	moistPctlst				3	ser1/MyA	rduino	moistA	naMin		231		231		INT	@moist	ŀ
Đ	wateringEnab	e			4	ser1/MyA	rduino	moistAr	naMax		1024				INT	@moist	ł
Đ	valveOpen				5	ser1/MyA	rduino	moist	Pctlst		94.9559				FLOAT	@moist	F
Þ	dryAlarm				6	ser1/MyA	rduino	moistP	ctSoll		50				FLOAT	@moist	F
Đ	timerFsm				7	ser1/MyA	rduino	stateCu	urrent		1					@state	2
(H)	dryAlarmTime	r			8	ser1/MyA	rduino	stateStrin	ng[029]	STATE	_1_VALVE_0	CLOSED			STRING	@states	51
1	stateCurrent		~		9												~
<		,		<												>	
ln Ti	ime				Node			Mes	sage								^
000 3,	/08/2021 14:5	4:15.48	37		ser1/COM	133/MyArd	uino	Clos	ing comm	unications	s port						
001 3,	/08/2021 14:5	5:27.34	0		ser1/CON	A33/MyArd	uino	Ope	ning serial	port [COI	M33] OK						~
<																`	

Fig. 2.7. Watering enabled turned on, soil wet enough.

 $\Rightarrow$  As watering is enabled now, the FSM changes to the state "STATE\_1\_VALVE\_CLOSED".

Ne	w Open	<b>₽</b> Save	B Print	Сору	K Cut	Deste Paste	0fflin	ne Setup()	کے Loop()	Write all	► Continue	• brkpt	⊠ brkpt		
	+ moistAna + moistAnaMir	r	^		In	terface/A	rd. ID	Variable I	name	Ac	tual value		Write value	Format	
	+ moistAnaMa	x		1	s	er1/MyArd	luino	wateringE	nable	0b	0000'0001		1	BIT	@watering
	+ moistPctSoll			2	s	er1/MyArd	luino	moistA	na		278			INT	@moistAna
b	+ moistPctlst			3	s	er1/MyArd	luino	moistAna	Min		231		231	INT	@moistAna
0	wateringEnal	ole		4	s	er1/MyArd	luino	moistAna	Max		1024			INT	@moistAna
	•valveOpen			5	s	er1/MyArd	luino	moistPc	tlst		94.0731			FLOAT	@moistPct
1	+ dryAlarm			6	s	er1/MyArd	luino	moistPct	Soll		50			FLOAT	@moistPct
	• timerFsm			7	S	er1/MyArd	luino	stateCur	rent		1				@stateCurr
ľ	dryAlarmTim	er		8	s	er1/MyArd	luino	stateString	[029]	STATE_1	VALVE_CLC	OSED		STRING	@stateStrin
ľ	+ stateCurrent			9	S	er1/MyArd	luino	dryAlarm	Timer		0			INT	@dryAlarm
	+ valve limeOp	benMSec		10	s	er1/MyArd	luino	valveTimeOp	enMSec		10000		10000	INT	@valveTim
ľ	+ day Time Aler			11	S	er1/MyArd	luino	valveTimeHo	dMSec		60000		10000	INT	@valveTim
ľ	+ totalValveTin	nor		12	S	er1/MyArd	luino	timerFs	m		0			INT	@timerFsm
ľ		lier		13	s	er1/MyArd	luino	valveOp	ben		0			INT	@valveOpe
	+ stateString[0	291		14	S	er1/MyArd	luino	totalValve	Timer		0				@totalValv
	Statestinigto		~	<											
	Time		N	lode			N	lessage							
	3/08/2021 15:	10:57.013	S	er1/COM33	/MyArc	luino	S	tate change t	o: STATE_0	_VALVE_W	ATER_OFF				
1	3/08/2021 15:	10:57.013	S	er1/COM33	/MyArc	luino	N	lew step = ST	TATE_1_VA	LVE_CLOSE	D				
5	3/08/2021 15:	10:57.013	S	er1/COM33	/MyArc	luino	S	tate change t	to: STATE_1	1_VALVE_CI	OSED				
F.	3/08/2021 15:	15:24.922	S	er1/COM33	/MyArc	luino	[F	Row 14] Writi	ng variable	e totalValve	Timer=0 ()	comple	eted.		

#### ✓ Monitor the variables as indicated in the table below.

Fig. 2.8. Watering enabled turned on, soil wet enough.

- $\Rightarrow$  The state machine is in the state "STATE\_1\_VALVE\_CLOSED" [line 8].
- $\Rightarrow$  The valve is closed [line 13].
- ⇒ totalValveTimerMSec, the total watering time since the boot of the Mega, is 0 [line 12].
- $\Rightarrow$  the value of *valveTimeOpenMSec* (the setpoint of the time the valve is open) is 10 seconds [11].
- ⇒ the value of *valveTimeHoldMSec* (the setpoint of the time the valve is closed to let the water settle) is 60 seconds [11].

#### 2.6 Simulate drought

✓ Remove the sensor from the water.

ا 🚰 File	WawiLib-PC [C:\l	Jsers\Joł	ni\Docur	ments	NoName	e.Wif*]-	[C:\Users\.	Johi\Doc	uments\We	llData\Wel	l1.bak.Wvf	f*]			-		×	
Ne	w Open	Save	👼 Print	C	© Сору	X Cut	® Paste	0fflin	e Setup()	Toop()	Write al	▶ I Continu	e brkpt	⊠ brkpt				
1	+ moistAna			^		In	terface/A	rd. ID	Variable	name	A	ctual valu	e	Write value	Format			^
	+ moistAnaMax	ć			1	se	er1/MyArd	uino	wateringE	nable	Ob	000'000	1	1	BIT	@waterir	ngEna	
	+ moistPctSoll				2	Se	er1/MyArd	uino	moistA	na		1023			INT	@moistA	na=(	
	• moistPctIst				3	Se	er1/MyArd	uino	moistAn	Min		231		231	INT	@moistA	naM	
	wateringEnab	le			4	s	er1/MyArd	uino	moistAna	Max		1024			INT	@moistA	naM	
	talveOpen				5	S	er1/MyArd	uino	moistPo	tlst		0.126106			FLOAT	@moistP	ctlst=	
	EdryAlarm				6	S	er1/MyArd	uino	moistPc	Soll		50			FLOAT	@moistP	ctSol	
	timerFsm •				7	Se	er1/MyArd	uino	stateCur	rent		2				@stateCu	Jrren	
	dryAlarmTime	er			8	se	er1/MyArd	uino	stateString	[029]	STATE	_2_VALVE_	OPEN		STRING	@stateSt	ring=	
	* stateCurrent	1.10			9	Se	er1/MyArd	uino	dryAlarm	Timer		1			INT	@dryAla	mTir	
	+ valve limeOp	enMSec			10	S	er1/MyArd	uino v	alveTimeOp	enMSec		10000		10000	INT	@valveT	meO	
	and a valve i imeno				11	S	er1/MyArd	uino 🕔	alveTimeHe	oldMSec		60000		10000	INT	@valveTi	meH	
	# total ValveTim	nvisec			12	S	er1/MyArd	uino	timerF	sm		4052			INT	@timerF	sm=C	
	Array	lei			13	S	er1/MyArd	uino	valveO	ben		1			INT	@valveO	pen=	
	+ stateString[0.	.291			14	Se	er1/MyArd	uino	totalValve	Timer		5948				@totalVa	lveTi 🗤	~
	States an igtor			~ <													>	
In	Time			Node	e			M	essage								1	~
043	3/08/2021 15:1	0:57.013		ser1/	/COM33/	MyArd	uino	Sta	ite change	to: STATE_	1_VALVE_C	LOSED						
044	3/08/2021 15:1	5:24.922		ser1/	/COM33/	MyArd	uino	[Re	w 14] Writi	ng variabl	e totalValv	eTimer=0	() comple	eted.				
045	3/08/2021 15:1	8:08.245	2	ser1/	/COM33/	MyArd	uino	Ne	w step = S	TATE_2_VA	LVE_OPEN							
046	3/08/2021 15:1	8:08.245		ser1/	/COM33/	MyArd	uino	Sta	te change	to: STATE_	2_VALVE_C	PEN					`	~
<																	>	
				Loop()	Autow	rite on	No reco	orders ac	ive ser1=	MyArduin	o=COM33	3/115200,8	,N,1,AVR	[ITF_LOOP] ms	g.ok/tot:	113062/1	13077	

- ✓ The state machine is in the state "STATE\_2\_VALVE\_OPEN" [line 8].
- ✓ The valve is open [line 13].
- ✓ totalValveTimerMSec , the total watering time since the boot of the Mega, is 5948 msec [line 12].
- ✓ the value of *valveTimeOpenMSec* (the setpoint of the time the valve is open) is 10 seconds [11].
- ✓ the value of *valveTimeHoldMSec* (the setpoint of the time the valve is closed to let the water settle) is 60 seconds [11].
- ✓ Put the sensor back in the water.

Ne	w Open	Save	en Print	Сору	K Cut	Paste	Offli	ne Setup()	Toop()	Write all Continue	brkpt	⊠ brkpt		
	+ moistAna		^		Int	terface/A	rd. ID	Variable r	ame	Actual value		Write value	Format	
	+ moistAnaMa	IX		1	se	er1/MyArd	uino	wateringEr	nable	060000'0001		1	BIT	@watering
	+ moistPctSoll			2	se	er1/MyArd	uino	moistAr	na	275			INT	@moistAn
	+ moistPctIst			3	se	er1/MyArd	uino	moistAna	Min	231		231	INT	@moistAn
	wateringEna	ble		4	se	er1/MyArd	uino	moistAna	Max	1024			INT	@moistAn
	■valveOpen			5	se	er1/MyArd	uino	moistPct	tlst	94.4514			FLOAT	@moistPct
	to dryAlarm			6	se	er1/MyArd	uino	moistPct	Soll	50			FLOAT	@moistPct
	🗄 timerFsm			7	se	er1/MyArd	uino	stateCurr	ent	3				@stateCur
	dryAlarmTim	ner		8	se	er1/MyArd	uino	stateString	029]	STATE_3_VALVE_CLOS	ED_HOLD		STRING	@stateStrin
	* stateCurrent			9	se	er1/MyArd	uino	dryAlarmT	imer	0			INT	@dryAlarm
	+ valveTimeO	benMSec		10	se	er1/MyArd	uino	valveTimeOp	enMSec	10000		10000	INT	@valveTim
	+ valve limeHo			11	se	er1/MyArd	uino	valveTimeHo	IdMSec	60000		10000	INT	@valveTim
	total/ab/aTi	mivisec		12	se	er1/MyArd	uino	timerFs	m	13364			INT	@timerFsm
	Array	ner		13	se	er1/MyArd	uino	valveOp	en	0			INT	@valveOp
	+ stateString[0	29]		14	se	er1/MyArd	uino	totalValvel	limer	20003				@totalValv
			~	<										>
<b>1</b>	Time		N	lode			N	lessage						
53	3/08/2021 15:	19:18.238	s	er1/COM33	/MyArd	uino	N	lew step = ST	ATE_2_VA	ALVE_OPEN				
54	3/08/2021 15:	19:18.238	s	er1/COM33	/MyArd	uino	S	tate change t	o: STATE_	2_VALVE_OPEN				
55	3/08/2021 15:	19:28.234	s	er1/COM33	/MyArd	uino	N	lew step = ST	ATE_3_VA	ALVE_CLOSED_HOLD				
56	3/08/2021 15:	19:28.234	s	er1/COM33	MyArd	uino	S	tate change t	o: STATE_	3_VALVE_CLOSED_HOL	D			

Fig. 2.10. xxx.

➡ The cycle continues until timerFsm is 0 and then the FSM changes to "STAE\_3\_VALVE\_CLOSED\_HOLD" ⇒ The variable "valveOpen" changes to 0 and stays there.

#### 2.7 Alarm generation

- ✓ Remove the sensor from the water.
- ✓ Look at the variables as indicated in the table below.

New	Den Save	en Print	Сору	X B 1 Cut Paste Off	រ line Setup()	Toop()	Write all Continue brkpt	⊠ brkpt		
+	moistAna	^		Interface/Ard. ID	Variable n	ame	Actual value	Write value	Format	Var
+	moistAnaMax		1	ser1/MyArduino	wateringEr	able	1		INT	@wateringEnable=0x04
+	moistPctSoll		2	ser1/MyArduino	dryTimeAlarr	nMSec	1800000		INT	@dryTimeAlarmMSec=
+	moistPctIst		3							
÷	wateringEnable		4							
ŧ	valveOpen		5	ser1/MyArduino	dryAlarmT	imer	0		INT	@dryAlarmTimer=0x05
+	dryAlarm		6	ser1/MyArduino	dryAları	n	1	1	INT	@drvAlarm=0x04D2 [1
+	timerFsm dryAlarmTimer	~	<							>
In T	Time	I	Node		Message					^
098 3	3/08/2021 15:26:03.849	5	er1/COM33/	MyArduino	New step = ST	ATE_O_VAL	VE_WATER_OFF			
099 3	3/08/2021 15:26:03.849	S	er1/COM33/	MyArduino	State change to	: STATE_0	_VALVE_WATER_OFF			
100 3	3/08/2021 15:26:03.849	s	er1/COM33/	MyArduino	New step = ST	ATE_1_VAL	VE_CLOSED			
101 3	3/08/2021 15:26:03.849	5	er1/COM33/	MyArduino	State change to	: STATE_1	_VALVE_CLOSED			
102 3	3/08/2021 15:26:03.850	5	er1/COM33/	MyArduino	New step = ST	ATE_2_VAL	VE_OPEN			
	3/08/2021 15:26:03.850	5	er1/COM33/	MyArduino	State change to	: STATE_2	_VALVE_OPEN			
103 3	3/08/2021 15:26:13.843	5	er1/COM33/	MyArduino	New step = ST	ATE_3_VAL	VE_CLOSED_HOLD			
103 3 104 3	.,,									

Fig. 2.11. Dry soil alarming.

- ⇒ The timer *dryAlarmTimer de*creases.
- $\Rightarrow$  As it reaches 0, the status of *dryAlarm* goes high.

#### 2.8 WawiLib as HMI

If you want to control the application remotely, you can use WawiLib to control the variable *wateringEnable* to turn watering on and off. The variable *moistPctSoll* can be used to control the setpoint and *moistPctIst* to observe the actual value. *totalValveTimer* contains the total time the water was running.

Observing and controlling the values of the variables explained in the previous paragraph gives you full control over your application. In the future, SylvesterSolutions will release a DLL that enables you to manipulate these variables using an application on the PC you have written yourself.

# 3 "WawiWaterSensorValve" data recording demo

# 3.1 Continuous recording of the moisture level

✓ Add a data recorder REC1 to *moistPctIst* and *moistPctSoll* (time-based 3 seconds for demo purposes):

Jata rec	ording setting	gs					×
Data reco	order name:	REC1					
File pro	perties Reco	ord details	Disk usage and file size	limitation			
Filenar	me and direct	ory		Data file form	at		
Filena	me: GreenH	ouseDemo.x	ml	O csv: comm	a separated values	;	
Direct	ory: C:\Users	;\Johi\Docur	nents	• xml: extens	sible markup langu	lage	
When	going online	on Arduino:		🔿 xlsx: Excel/	LibreOffice compa	tible s	preadsheet
Ove	erwrite currer	ıt data file		CSV separator	\t=tab)		
	pend <mark>new d</mark> at	a records to	current data file				
⊖ Sta	rt with new d	ata file (add	start date and time to fi	lename)			
Configu	ired data reco	orders					
	File mode	Time hase	File	Dir	Add record type		
Name	i ne mode	Time base	The	DI	Add record type	A	Add
Name REC1	OVERWRITE	3 sec	GreenHouseDemo.xml	C:\Users\Johi\Documents	no	ye	Add Remove
Name REC1	OVERWRITE	3 sec	GreenHouseDemo.xml	C:\Users\Johi\Documents	no	Ye	Add Remove Update
REC1	OVERWRITE	3 sec	GreenHouseDemo.xml	C:\Users\Johi\Documents	no	Xi yi	Add Remove Update Clear list

Fig. 3.1. Define a new data recorder.

Data recording settings				×
Data recorder name: REC1				
File properties Record det	tails Disk usage and file size limi	tation		
Recording triggers Record time based (for Recording interval (sec):	r change based: modify properties	of variable in main table)		
Data record type	based/change based)			
Data record timestamp set Add date Add time Add milliseconds (appr	ttings Add date (UTC) Add time (UTC) rox.) Add relative timestamp	(elapsed time)		
Configured data recorders				
Name File mode Time	base File	Dir	Add record type	Add
REC1 OVERWRITE 3 sec	GreenHouseDemo.xlsx.csv	C:\Users\Johi\Documents	no	Remove
				Update
<			>	Clear list
	OK Can	cel Default parar	neters	5 Se

Fig. 3.2. New variable data recorder 3 sec timebase.

- ✓ Disable protocol tracing in the output window (click right on the output window).
- ✓ Disable .print() tracing in the output window.
- ✓ Enable recorder tracing in the output window.
- $\checkmark$  Slowly pull the sensor out of the water and put it back in.
- ✓ Repeat this action multiple times.
- ✓ Select the variables *moistPctIst* and *moistPctSoll* for recording (add text REC1 to the recorder column).

) Loop() Write a	1					
Actual value	Write value	Format	Recorde	r	1^	•
1023		INT		@moistAna=0x03DE [2	byte] x 1 -	
200	200	INT		@moistAnaMin=0x020/	A [2 byte]:	
1024		INT		@moistAnaMax=0x020	C [2 byte]	
0.121361		float	REC1	@moistPctIst=0x03DA [	[4 byte] x i	L
50	60	float	DEC1			
ALVE_CLOSED_HOL		string	A	vailable interfaces	· /	
10		INIT	A	vailable data recorders	>	
0		INT	Di	splay formats	>	Т
			<ul> <li>Of</li> </ul>	ffline	~	
			Se	tup()		
			Lo	0000	^	
ange to:VALVE_OPEN						
ange to:VALVE CLOSE	DHOLD		W	rite row C	Ctrl+W	

Fig. 3.3. Select data recorder rect 1 for *moistPctIst* and *moistPctSoll*.

New	Dpen	<b>⊌</b> Save	e Print	ору (	🔏 🖻 Cut Paste	Offli	ine Se	To etup() Lo	oop() Write a	▶ Il Continue	• brkpt	• brkpt	
	moistAna	Min	^		Interface/	Ard. ID	Variat	ole name	Actual value	Write value	Format	Recorder	Variat
	moistAna	Max		1	ser1/MyA	rduino	moi	stPctlst	100.967		FLOAT	REC1	@moistPctIst=0
	moistPctS	oll		2	ser1/MyA	rduino	mois	tPctSoll	50		FLOAT	REC1	@moistPctSoll=
Đ	moistPctl	st		3									
Đ	wateringE	nable		4									
4	valveOne	n	~ <										
I Ti	ime		Noc	de		٢	Message	9					
46 3,	/08/2021	15:32:27.059	REC	1		r	rec(o.t.):	3/08/2021	15:32:27 0.138	3123 50			
7 3,	/08/2021	15:32:30.061	REC	:1		r	rec(o.t.):	3/08/2021	15:32:30 0.138	3123 50			
48 3,	/08/2021	15:32:33.058	REC	1		r	rec(o.t.):	3/08/2021	15:32:33 0.138	3123 50			
49 3,	/08/2021	15:32:36.059	REC	1		r	rec(o.t.):	3/08/2021	15:32:36 0.138	3123 50			
50 3,	/08/2021	15:32:39.057	REC	1		r	rec(o.t.):	3/08/2021	15:32:39 0.138	3123 50			
51 3,	/08/2021	15:32:42.060	REC	1		r	rec(o.t.):	3/08/2021	15:32:42 109.9	945 50			
52 3,	/08/2021	15:32:45.064	REC	1		r	rec(o.t.):	3/08/2021	15:32:45 106.6	53 50			
53 3,	/08/2021	15:32:48.017	REC	:1		r	rec(o.t.):	3/08/2021	15:32:48 96.40	088 50			
54 3,	/08/2021	15:32:51.022	REC	1		r	rec(o.t.):	3/08/2021	15:32:51 0.138	3123 50			
55 3,	/08/2021	15:32:54.030	REC	:1		r	rec(o.t.):	3/08/2021	15:32:54 105.3	387 50			
56 3,	/08/2021	15:32:57.037	REC	1		r	rec(o.t.):	3/08/2021	15:32:57 102.6	524 50			
57 3,	/08/2021	15:33:00.039	REC	1		r	rec(o.t.):	3/08/2021	15:33:00 0.138	3123 50			
58 3,	/08/2021	15:33:03.044	REC	1		r	rec(o.t.):	3/08/2021	15:33:03 100.8	329 50		N	
59 3,	/08/2021	15:33:06.047	REC	.1		r	rec(o.t.):	3/08/2021	15:33:06 101.5	519 50		15	
50 3,	/08/2021	15:33:09.046	REC	.1		r	rec(o.t.):	3/08/2021	15:33:09 101.5	519 50			
61 3,	/08/2021	15:33:10.722	REC	1		V	write foo	oter: 3/08/	2021 15:33:11	ile closed: offli	ne		
62 3,	/08/2021	15:33:10.722	REC	1		c	closing	opened file	e C:\Users\Johi	\Documents\G	reenHous	eDemo.xlsx	.csv OK

Fig. 3.4. Varying values of *moistPctIst* and static values of *moistPctSoll*.

✓ Open the recorded .xlsx file in Microsoft Excel or LibreOffice calc:

6	<b>.</b>	<del>,</del> Green	ouseD Sig	n in 🖪 —	o x
Fi	le Hor Inser	Page Forn Da	ata Revie View D	Deve Help 🛛 Q Tell	me >
A1	~	: × 🗸	fx date	2	~
	А	В	С	D	E 🔺
1	date	time	moistPctlst (Fl	moistPctSoll (FLOAT)	
2	3/08/2021	15:32:18	0.138123	50	
3	3/08/2021	15:32:21	0.138123	50	
4	3/08/2021	15:32:24	0.138123	50	
5	3/08/2021	15:32:27	0.138123	50	
6	3/08/2021	15:32:30	0.138123	50	
7	3/08/2021	15:32:33	0.138123	50	
8	3/08/2021	15:32:36	0.138123	50	
9	3/08/2021	15:32:39	0.138123	50	
10	3/08/2021	15:32:42	109.945	50	
11	3/08/2021	15:32:45	106.63	50	
12	3/08/2021	15:32:48	964.088	50	
13	3/08/2021	15:32:51	0.138123	50	
14	3/08/2021	15:32:54	105.387	50	
15	3/08/2021	15:32:57	102.624	50	
16	3/08/2021	15:33:00	0.138123	50	
17	3/08/2021	15:33:03	100.829	50	
18	3/08/2021	15:33:06	101.519	50	
19	3/08/2021	15:33:09	101.519	50	
20	3/08/2021	15:33:11	file closed: off	line	
21					-
	Green	HouseDemo	xlsx +	•	►
Read	dy 🙀 Display S	Settings 🏼 🆽	I II		+ 100%

Fig. 3.6. Recored data file opened in Microsoft Excel.

 $\Rightarrow$  You can see the recorded values of the moisture setpoint and actual value.

# 3.2 Change based recording of the state machine status.

# ✓ Add a data recorder to record the values of *stateString* and *valveOpen* (change based):

	ording setting	js					×		
D <mark>ata rec</mark> o	order name:	REC2							
File pro	perties Reco	ord details	Disk usage and file size	limitation					
Filenar	ne and direct	ory		Data file forma	it				
Filena	me: GreenHo	ouseDemo.c	sv	🔘 csv: comma	• csv: comma separated values				
Direct	ory: C:\Users	\Johi\Docur	nents	O xml: extens	ible markup langu	lage			
When	going online	on Arduino:		🔿 xlsx: Excel/L	ibreOffice compa	tible	spreadsheet		
Over	erwrite curren	it data file		CSV separator (	\t=tab)				
	pend new dat	a records to	current data file						
Osta	rt with now d	ata filo (add	start data and time to fi	lonamo)					
Configu	File mode	Time base	File	Dir	Add record type	Δ.			
Configu Name REC1	File mode	Time base	File GreenHouseDemo.xlsx	Dir C:\Users\Johi\Documents	Add record type	A	Add		
Configu Name REC1 REC2	File mode OVERWRITE OVERWRITE	riders Time base 3 sec No	File GreenHouseDemo.xlsx GreenHouseDemo.csv	Dir C:\Users\Johi\Documents C:\Users\Johi\Documents	Add record type no no	Ai ye ye	Add Remove		
Configu Name REC1 REC2	File mode OVERWRITE OVERWRITE	Time base 3 sec No	File GreenHouseDemo.xlsx GreenHouseDemo.csv	Dir C:\Users\Johi\Documents C:\Users\Johi\Documents	Add record type no no	Aı ye ye	Add Remove Update		
Configu Name REC1 REC2	File mode OVERWRITE OVERWRITE	orders Time base 3 sec No	File GreenHouseDemo.xlsx GreenHouseDemo.csv	Dir C:\Users\Johi\Documents C:\Users\Johi\Documents	Add record type no no	A ye ye	Add Remove Update Clear list		

Fig. 3.7. Add a new data recorder REC2, disable time based recording.

Data recording settings												
Data recorder name: REC2												
File properties Record details Disk usage and file size limitation												
Recording triggers          Record time based (for change based: modify properties of variable in main table)         Recording interval (sec):												
Data record type Add record type (time based/change based)												
Data record timestamp settings       Add date       Add time       Add time       Add time       Add milliseconds (approx.)												
Configured data recorders												
Name File mode Time base File Dir Add record type A Add												
REC1 OVERWRITE 3 sec GreenHouseDemo.xml C:\Users\Johi\Documents no ye												
REC2 OVERWRITE No GreenHouseDemo.xml C:\Users\Johi\Documents no ye Update												
OK Cancel Default parameters												

Fig. 3.8. Add a new data recorder REC2, disable time based recording.

- ✓ Add REC2 as recorder for moistAna, *stateString, and sateCurrent*.
- ✓ Change the properties of *stateString* as indicated below:

2 Now	Open	Save	8 Print	Conv	ж Сut	Dasto	Offline	To Setup()	Toon(	Write all	Continue	• brknt	⊠ brkpt
Availabl	e vari ^	Jave	nter	face/Ard. II	Varia	ble name	Act	ual value	LOOP	Write value	Format	Recorder	bikpt
Sing	le va	1	ser1,	/MyArduino	water	ringEnable	gEnable 1			INT		@wateri	
1 mc	oistAr	2	ser1,	/MyArduino	m	oistAna	287				INT	REC2	@moist/
emoistAr 3 ser			ser1,	/MyArduino	state	eCurrent	1				INT	REC2	@stateC
± moistAr		4	ser1,	/MyArduino	stateS	tring[029]	STATE_1_VALVE_CLOSED				string	REC2	@stateS
1±mc	DistPc v	5			_	Variable p	roportion			×			
<b>C</b>	>	<				variable p	operties			~			>
n Time	e		1	Node		Timing	Recordin	g					
:					-	☑ Reco	rding ena	oled					
.0000()	Autowrite	on REC	[RECO		t=0	Change of variable triggers data recorder							98/11741
						Default	value if re	ead fails:	ERR				

Fig. 3.9. Change the properties of stateString so its changes trigger REC2.

- ✓ Enable "Display data recording" in the output window.
- ✓ Delete REC1.
- ✓ Go online, keep the sensor out of the water.
- ⇒ You can see the various state changes of the FSM recorded as they change. If the state does not change, there will be no record added to the recording file.

231	WawiLib-PC [C	:\Users\Joh	i\Docum	nents\No	Name.W	/if*]-[C:\Users\J	Johi\Doc	uments\We	IData\We	ell1.bak.Wvf*]		—		×
File	Edit Settings	Help												
e Ne	w Open	<b>⊌</b> Save	8 Print	Cop	y Ci	ut Paste	0fflin	e Setup()	ت Loop()	↓ Write all Continue	• brkpt brk	• kpt		
	ailable variable	es	^			Interface/Arc	d. ID	Variable	name	Actu	al value	١	Nrite value	e ( ^
	Single variab	le		1	I	ser1/MyArdu	iino	watering	Enable		1			
	+ moistAna			2	2	ser1/MyArdu	lino	mois	Ana		252			
	moistAnaM	lin		6	3	ser1/MyArdu	iino	stateC	irrent		3			
	the moistAnaM	ax		2	1	ser1/MyArdu	ino	stateStrin	g[029]	STATE_3_VAL	/E_CLOSED_HC	DLD		
	moistPctSo	I	~	<	-									>
In         Time         Node           000         3/08/2021         15:57:15.232         REC2           001         3/08/2021         15:57:15.232         REC2           002         3/08/2021         15:57:28.546         REC2           003         3/08/2021         15:57:28.546         REC2           004         3/08/2021         15:57:28.546         REC2           004         3/08/2021         15:57:28.546         REC2           005         3/08/2021         15:57:36.224         REC2           006         3/08/2021         15:57:46.185         REC2           007         3/08/2021         15:58:6.092         REC2           008         3/08/2021         15:59:07.464         REC2           010         3/08/2021         15:59:77.627         REC2           011         3/08/2021         15:59:21.789         REC2           012         3/08/2021         15:59:21.789         REC2						wr clc op wr ree ree ree ree ree ree ree ree ree	ite footer: 3 osing opene pening file C (ite header: ( c(o.c.): 3/08/ c(o.c.): 3/08/ c(o.c.): 3/08/ c(o.c.): 3/08/ c(o.c.): 3/08/ c(o.c.): 3/08/ ite footer: 3 osing opene	/08/2021 d file C:\L :\Users\Jc date time 2021 15:5 2021	15:57:15 file closed: of Jsers\Johi\Documents\GreenH moistAna (INT) stateCu 57:29 287 1 STATE_1_VA 57:36 1023 2 STATE_3_VA 58:46 1023 2 STATE_3_VA 58:46 1023 2 STATE_3_VA 58:56 1023 3 STATE_3_V 59:07 1023 2 STATE_3_VA 15:59:22 file closed: of Jsers\Johi\Documents\	fline GreenHouseDe JouseDemo.csv urrent (INT) sta LIVE_CLOSED ALVE_OPEN ALVE_CLOSED_ ALVE_CLOSED_ ALVE_CLOSED_H fline GreenHouseDe	emo.csv C v for oven iteString[( HOLD _HOLD HOLD emo.csv C	0K write OK −29] (STRIN	1G)	
<		Offline	Autour	ite on			7 cor1	- Mu Arduin		2/11E200 8 N 1 AVR (ITE		a ok /tot: 1	19022/119	>
_		Online	Autowr	ite on	REC2 [RE	CO_IDLEJ cht=	ser i	=iviyArduin		5/115200,0,10,1,AVR [11F	_OFFLINE] msg	g.ok/tot: I	10022/118	057

 $\Rightarrow$  Fig. 3.10. A change of the FSM state triggers the write of a new set of data to the disk.

- $\Rightarrow$  Once the PC reads a state change of the FSM, the data recorder writes the change to disk.
- $\Rightarrow$  You see that data is only recorded if the state of the FSM changes, otherwise noting is recorded.
- ➡ You see that other parameters referenced by REC2 are written on disk when the state string changes.
- ▷ In the same way as described before you can open the recorded datafile in LibreOffice Calc or Excel:

F	<b>ه</b> ۲ د		Gre	enHouseDe	emo.csv	- Excel	Sign	in 🖪	<b>D</b> —		>	<
Fi	le Home	Insert Pa	ige L For	mu Data	Review	View Devel	Help	Q Te	ell me	🖻 Sh	are	
A1	A1 • : $\times \checkmark f_x$ date											~
	А	В		С		D		E				
1	date	time	moistAn	a (INT)	stateC	Current (INT)	stateStr	ing[0-29]	] <mark>(</mark> STRING	)		
2	3/08/2021	15:57:29		287		1	STATE_	1_VALVE	CLOSED			
3	3/08/2021	15:57:36		1023		2	STATE_	2_VALVE	OPEN			
4	3/08/2021	15:57:46		991		3	STATE_	3_VALVE	CLOSED	HOLD		
5	3/08/2021	15:58:46		1023		2	STATE_	2_VALVE	OPEN			
6	3/08/2021	15:58:56		1023		3	STATE_	3_VALVE	CLOSED	HOLD		
7	3/08/2021	15:59:07		1023		2	STATE_	2_VALVE	OPEN			
8	3/08/2021	15:59:18		234		3	STATE_	3_VALVE	CLOSED	HOLD		
9	3/08/2021	15:59:22	file close	ed: offline								-
4	Gr	eenHouse	eDemo	+		1	4					
Rea	dy 💽		Lą	Display Set	tings		巴	<u> </u>		- + 1	00%	

Fig. 3.11. recorded data file opened in Microsoft Excel.

# 4 "WawiWaterSensorValve" Arduino software automation concepts

The objective of this part of the document is to document the main concepts used in the demo program. They come from the world of industrial automation.

## 4.1 Never put the main loop on hold or in delay

Essential in industrial automation applications is the main loop. This loop is run through many times per second. In this loop the logic is programmed to read the inputs, to calculate the result and to write the output signals.

For some applications, you would want to set an output signal OUT\_A when an input signal IN\_A comes high but after a time delay. In the same time, you would want to invert an input IN\_B to an output OUT\_B. The temptation is to use the delay instruction like this:

```
#define IO IN A 1
#define IO_OUT_A 2
#define IO IN B 11
#define IO_OUT_B 12
void setup() {
    pinMode(IO IN A, INPUT);
    pinMode(IO_OUT_A, OUTPUT);
    pinMode(IO_IN_B, INPUT);
    pinMode(IO OUT B, OUTPUT);
}
void loop()
{
    if (digitalRead(IO IN A))
    {
        delay(1000);
        digitalWrite(IO_OUT_A, HIGH);
    }
    else
        digitalWrite(IO OUT A, LOW);
    digitalWrite(IO_OUT_B, !digitalRead(IO_IN_B));
}
```

Fig. 4.1. using delay in a real time program is not a good idea.

The problem is that the code that inverts input B is put on hold by the delay(1000) statement.

What we need is some kind of timer logic that we can set and check and that runs in parallel or independently. One of the ways to create timer logic is to use a hardware timer based interrupt to decrement a variable that is holding a time value in an interrupt handler.

The hardware timer approach has its disadvantages: First of all, it will eat up (much) CPU time executing the interrupt handler. You also do not know at what line of the main loop the interrupt handler will kick in. Therefor you cannot assume that through the main loop, the timer will have the same value.

Last but not least the variable holding the time value needs to be declared "volatile". This prevents compiler optimizations that make code fail when variables are shared between interrupt handlers and main code. The volatile keyword tells the compiler to always use the same memory location for the variable instead of using registers at some places and memory at other places for efficiency reasons.

## 4.2 WawiTimer object

### 4.2.1 The concept "millis() without delay in a C++ object"

Arduino's have a function called *millis()*. Millis() returns the number of milliseconds since the Arduino board was booted. If we keep track of the value of millis(), it is possible to create a kind of timer "object" without blocking the main loop.

Suppose, we iterate through the main loop and each time we iterate, we store the value of millis().

By calculating the difference between the current and the previous value of millis(), we know the amount of time that has elapsed since the last pass.

Now, if we keep track of all the time intervals elapsed we know the total time elapsed between an event and the current moment in time. Doing so opens the way for a timer object.

So, first you set the object to a value (10.000 msec), each scan the value of 10 sec is decremented by the time elapsed since the previous scan. We do this until the value of the time counter reaches 0.

Each time we look also at the value of the internal time counter, as long as it is bigger than 0, we do nothing and once is it zero, we start some kind of job or a task. This way we can create a timer functionality without holding the loop. Therefor other jobs that are also managed by the program do not need to be put on hold or frozen.

If you put this functionality in an encapsulated object, you get the object called "WawiTimer". The object has its own files WawiTimer.cpp and WawiTimer.h.

#### 4.2.2 Short look under the hood.

The first thing you do is

WawiTimer timerFsm;

Fig. 4.3. WawiTimer class members (WawiTimer.h).

Once you do this a number of member variables are created as declared in the declaration of the C++ object class in the header.

```
class WawiTimer
{
private:
    unsigned long m_tPrev;
    bool m_activePrev;
public:
    WawiTimer(bool incremental = false);
    unsigned long long m t;
    bool m_incremental;
public:
    bool isZero() const { return (m_t == 0); };
    void setMs(unsigned long long valueMs) { m_t = valueMs; m_tPrev = millis(); };
    void setMsAct(unsigned long valueMs) { m_tPrev = valueMs; };
    unsigned long long getValue() const { return m_t; };
    void loop(bool active = true);
    operator bool() const { return (m_t == 0); };
};
```

Once you call timerFsm.loop() (that is a public member function) the whole principle as declared in the \$4.2.1. is executed.

```
void WawiTimer::loop(bool active)
{
   unsigned long m, delta;
   if (!m_activePrev && active)
   {
       m_tPrev = millis();
       m_activePrev = active;
       return;
   }
   // calculate delta time between last loop and this loop;
   if (m_activePrev)
   {
       m = millis();
       delta = m - m_tPrev;
       m_tPrev = m;
   }
   // incrementing timer counter:
   if (m_incremental && m_activePrev)
   {
       m_t = m_t + delta;
       else
           m_t = 0xFFFFFFFFFFFFFF;
   }
   // decrementing timer counter:
   if (!m_incremental && m_activePrev)
   {
       if (m t >= delta)
           m_t = m_t - delta;
       else
           m_t = 0;
   }
   m activePrev = active;
1
```

Fig. 4.4. WawiTimer internal loop function.

So when the Arduino executes:

timerFsm.isZero(), it executes the code marked in yellow below.

```
class WawiTimer
{
private:
    unsigned long m_tPrev;
    bool m_activePrev;
public:
    WawiTimer(bool incremental = false);
    unsigned long long m_t;
    bool m_incremental;
public:
    bool isZero() const { return (m_t == 0); };
    void setMs(unsigned long long valueMs) { m_t = valueMs; m_tPrev = millis(); };
    void setMsAct(unsigned long valueMs) { m_tPrev = valueMs; };
    unsigned long long getValue() const { return m_t; };
    void loop(bool active = true);
    operator bool() const { return (m_t == 0); };
};
```



The nice thing about this is that the whole millis() thing is encasulated, (hidden) by the object so it does not trouble the structure the code as many implementations that use millis() directly do.

# 4.3 The finite state machine concept

### 4.3.1 Introduction

I have been writing software for more than 30 years now and I have been the witness of epic discussions of good and bad software. I have seen bad software work and good software fail.

Some time ago I read a small book from Mark Buelens, a professor at the Vlerick Management school about decision making. The most important lesson he learned was: do focus on the quality of the decision process. The result of a decision can turn out good or bad because a lot of factors are involved, but whatever you do: focus on the quality of the process. My experience with software is similar, if you use methods that reduce the risk of failure you will get better results.

If you want to write a program that does some kind of sequencing, you can use a lot of Boolean logic to implement the program. But if you are not careful, you risk to create situations where your logic is blocked and hangs. Last minute modifications or small modifications create high risk.

### 4.3.2 Finite state machine concept

A finite state machine is a concept where the machine can only have a limited number of states. It can only change from one defined state to another defined state using transitions. These transitions are well defined and only influence the transition from step A to B and nothing else.



Fig. 4.6. finite state machine of watering application.

This means if you want to implement additional transitions, you only modify a small part of the code that is isolated from other transitions. This makes the risk of destroying the code logic much smaller than in code that has been built without this type of logic.

In the figure 4.6. you see a visual representation of the finite state machine for the WawiWaterSensorValve program. If the program is not active, the FSM is in the state *STATE\_0\_VALVE\_WATER\_OFF*.

In the state *STATE\_0\_VALVE\_WATER\_OFF*, the condition *wateringEnable* == true triggers the transition from *STATE\_0\_VALVE\_WATER\_OFF* to *STATE\_1\_VALVE\_CLOSED*.

In the state *STATE\_1\_VALVE\_VALVE\_CLOSED*, the condition *moistPctSoll > moistPctIst* (actual value of moisture measured is below the setpoint) triggers the transition from *STATE\_1\_VALVE\_WATER\_OFF* to *STATE\_2\_VALVE\_OPEN*.

In the state *STATE\_2\_VALVE\_VALVE\_OPEN*, the condition timerFsm.isZero() (count down timer has elapsed) triggers the transition to *STATE\_3\_VALVE\_CLOSED\_HOLD*.

Other transitions in the diagram work in the same way.

### 4.3.3 Finite state machine implementation

In C++ programming, an enumeration type (or an enumerator class) is a data type that ideal to represent the states of an FSM. This approach is type safe compared to an approach with an integer number that could obtain an illegal value because of a programming error.

```
// finite state machine variables to control the valve:
// (a finite state machine is a construct that can only have a limited number of
states)
enum class FsmState { STATE_0_VALVE_WATER_OFF = 0, STATE_1_VALVE_CLOSED = 1,
STATE_2_VALVE_OPEN = 2, STATE_3_VALVE_CLOSED_HOLD = 3 };
FsmState stateCurrent, statePrev{ FsmState::STATE_0_VALVE_WATER_OFF };
const char* stateStrings[] = { "STATE_0_VALVE_WATER_OFF", "STATE_1_VALVE_CLOSED",
"STATE_2_VALVE_OPEN", "STATE_3_VALVE_CLOSED_HOLD" };
char stateString[30];
bool firstScanNewState;
```

Fig. 4.7. define a enumerator to represent the states of the finite state machine.

If we want to display the actual value in WawiLib, we can use the variable *stateCurrent* and visualize it as an integer number. This is a poor solution as the numbers of the states do not give that much information.

Much better is to show the states of the FSM as text. To do the conversion to a string, I created an array of char\* strings *stateStrings[]* and a variable *stateString* that will contain the actual value of the state machine in plain text.

*Strcpy* copies one of the strings in *stateStrings*[] into *stateString*. The appropriate array index is obtained by using the *stateCurrent* as an integer. (int) is used to cast the Enumto an array index. (fig 4.8.)

// report: translate stateCurrent into a string and put it in stateString: strcpy(stateString, stateStrings[(int)stateCurrent]);

Fig. 4.8. Convert enum into string containing the actual step.

The code in fig. 4.9. shows the implementation of the FSM itself. The case state using the enum *stateCurrent* makes sure that the code remains transparent. The use of the timer concept explained in the previous chapter is used to implement timer functions.

```
//-----
                                    -----
// helper to transfer FSM to other state:
void fsmGoTo(FsmState newState)
{
   stateCurrent = newState;
   WawiSrv.println((String)"New step = " + stateStrings[(int)stateCurrent]);
}
//-----
// finite state machine:
void loopFsmStateLogic()
{
   do
   {
       // Move from one step to another if conditions are right:
       switch (stateCurrent)
       {
       case FsmState::STATE_0_VALVE_WATER_OFF:
           if (wateringEnable)
               fsmGoTo(FsmState::STATE_1_VALVE_CLOSED);
           break;
       case FsmState::STATE_1_VALVE_CLOSED:
           if (moistPctSoll > moistPctIst)
           {
              WawiSrv.wawiBreak(1, "Break when soil is to dry!");
              fsmGoTo(FsmState::STATE 2 VALVE OPEN);
           }
           if (!wateringEnable)
              fsmGoTo(FsmState::STATE_0_VALVE_WATER_OFF);
           break;
       case FsmState::STATE_2_VALVE_OPEN:
           if (firstScanNewState)
           {
              WawiSrv.wawiBreak(2, "Break entering step 3.");
              timerFsm.setMs(valveTimeOpenMSec);
           }
           if (timerFsm.isZero())
               fsmGoTo(FsmState::STATE_3_VALVE_CLOSED_HOLD);
           if (!wateringEnable)
              fsmGoTo(FsmState::STATE_0_VALVE_WATER_OFF);
           break;
       case FsmState::STATE_3_VALVE_CLOSED_HOLD:
           if (firstScanNewState)
           {
              WawiSrv.wawiBreak(3, "Break entering step 4.");
              timerFsm.setMs(valveTimeHoldMSec);
           if (timerFsm.isZero())
               fsmGoTo(FsmState::STATE_0_VALVE_WATER_OFF);
           break;
       }
       firstScanNewState = (stateCurrent != statePrev);
       // report: translate stateCurrent into a string and put it in stateString:
       strcpy(stateString, stateStrings[(int)stateCurrent]);
       // display state change of the FSM in the output window:
       if (firstScanNewState)
       {
           statePrev = stateCurrent;
           WawiSrv.print("State change to: ");
```

```
WawiSrv.println(stateString);
        }
        while (firstScanNewState);
}
```

Fig. 4.9. the actual finite state machine implementation using a switch and case statements.

Each time the FSM goes from one state to another, the variable *firstScanNewState* is set. Upon entry of the Case section related to the new state, firstScanNewState is used to initialize timerFsm with the target value.

Then, the program keeps circulating through the case statement related to the current step loop after loop. This until (for example) *timerFsm.isZero()* returns true.

If a transfer condition becomes true, the FSM changes state to a new active state using the function *fsmGoTo()*.

#### 4.3.4 Output control using finite state machines.

The last step is to set or clear the Arduino outputs based on the state of the FSM.

```
//-----
// finite state machine:
void loopFsmStateToOutputs()
{
   // Move from one step to another if conditions are right:
   switch (stateCurrent)
   {
   case FsmState::STATE_0_VALVE_WATER_OFF:
   case FsmState::STATE_1_VALVE_CLOSED:
   case FsmState::STATE_3_VALVE_CLOSED_HOLD:
      valveOpen = LOW;
      break;
   case FsmState::STATE 2 VALVE OPEN:
      valveOpen = HIGH;
      break;
   }
```

Fig. 4.10. FSM deterimnes the status of the IO's.

In the same way as for the transitions, you can use a switch() statement to control the states of the FMS output. Putting the control of the outputs in a separate function keeps the code clean and easy to manage.

### 4.3.5 Other parts

The completing part of the application is some code to implement an alarm that will be triggered if the moisture level is too low for a period of time. If you have been able to follow until now, the code for the alarming will have no surprises. The code that drives WawiLib has been explained in the WawiLib "Getting Started" demos.

# 5 FURTHER READING

This demo demonstrates how to implement automation logic using a finite state machine and to test, debug and control the application using WawiLib. A very efficient concept of timers was introduced that does not block the main loop. A finite state machine (FSM) was used for analysis of the application. The FSM was translated in to Arduino C code using enums.

I hope you enjoyed this demo. Visit us on <u>www.sylvestersolutions.com</u> for the other demos.

# 6 APPENDIX: CODE LISTING

```
/*
* Project Name: WawiWaterSensorValve
* File: WawiWaterSensorValve.ino
* Description: demo file library for WawiSerialUsb libary.
* Reads water sensor, and contols water valve based on moisture level.
* Generates an alarm if moisture is below low level for too long
* Moisture level control can be turned on and off
* All parameters can be modified and observed using WawiLib
* Use programming USB port to make connection with the Arduino board.
* This software is provided as is without warranty of any kind
* Author: John Gijs.
* Created March 2020
* Updated August 2021
* More info: www.sylvestersolutions.com
* Technical support: support@sylvestersolutions.com
* Additional info: info@sylvestersolutions.com
*/
#include <WawiSerialUsb.h>
#include "WawiTimer.h"
// comment next line out (use //) if you do not have IO present and want to use
WawiLib to simulate IO
#define ENABLE_IO 1
#define VALVE_IO 13
#define DRY_ALARM_IO 13
//-----
// variables related to the actual moisture of the soil:
int moistAna = 0;  // analog input
int moistAnaMin = 300;  // scaling low limit
int moistAnaMax = 1024;  // scaling high limit
float moistPctIst = 0.0; // actual value in %
float moistPctSoll = 50.0; // low limit to start watering in %
//-----
// wateringEnable watering functionality is ON or OFF:
bool wateringEnable = false;
// actual state of watering valve itself and the alarm output:
bool valveOpen = false;
bool dryAlarm = false;
WawiTimer timerFsm;
WawiTimer dryAlarmTimer();
WawiTimer totalValveTimer(/*incrementing */ true);
//-----
// alarm delay time:
unsigned long long dryTimeAlarmMSec = 1800000;
//-----
// the water needs some time to settle before the moisture reaches the sensor
// so the valve needs to pulsate in order for the application to work properly:
long long valveTimeOpenMSec = 10000;
long long valveTimeHoldMSec = 60000;
//-----
// finite state machine variables to control the value:
```

```
// (a finite state machine is a construct that can only have a limited number of
states)
enum class FsmState { STATE 0 VALVE WATER OFF = 0, STATE 1 VALVE CLOSED = 1,
STATE_2_VALVE_OPEN = 2, STATE_3_VALVE_CLOSED_HOLD = 3 };
FsmState stateCurrent, statePrev{ FsmState::STATE_0_VALVE_WATER_OFF };
const char* stateStrings[] = { "STATE_0_VALVE_WATER_OFF", "STATE_1_VALVE_CLOSED",
"STATE_2_VALVE_OPEN", "STATE_3_VALVE_CLOSED_HOLD" };
char stateString[30];
bool firstScanNewState;
//-----
// the wawilib part:
WawiSerialUsb WawiSrv;
void wawiVarDef()
{
   WawiSrv.wawiVar(moistAna);
   WawiSrv.wawiVar(moistAnaMin);
   WawiSrv.wawiVar(moistAnaMax);
   WawiSrv.wawiVar(moistPctSoll);
   WawiSrv.wawiVar(moistPctIst);
   WawiSrv.wawiVar(wateringEnable);
   WawiSrv.wawiVar(dryTimeAlarmMSec);
   WawiSrv.wawiVar(valveTimeOpenMSec);
   WawiSrv.wawiVar(valveTimeHoldMSec);
   WawiSrv.wawiVar(valveOpen);
   WawiSrv.wawiVar(dryAlarm);
   WawiSrv.wawiVar(stateCurrent);
   WawiSrv.wawiVarTimer(timerFsm);
   WawiSrv.wawiVarTimer(dryAlarmTimer);
   WawiSrv.wawiVarTimer(totalValveTimer);
   WawiSrv.wawiVarArray(stateString);
}
//-----
_ _ _ _ _ _ _
void setupIO()
{
   pinMode(VALVE_IO, OUTPUT);
   pinMode(DRY_ALARM_IO, OUTPUT);
}
.
//-----
_ _ _ _ _ _ _
void setupWawiLib()
{
   WawiSrv.begin(wawiVarDef, Serial, "MyArduino");
   WawiSrv.wawiBreakDisable();
}
//-----
_ _ _ _ _ _ _
void setup()
{
   Serial.begin(115200);
   setupWawiLib();
   setupIO();
}
```

```
//====
            _____
_____
// scaling and limiting functions for analog I/O processing:
int limit(int v, int min, int max)
{
   if (v < min) return min;</pre>
   if (v > max) return max;
   return v;
}
float scale(int v, int vMin, int vMax, float lo, float hi)
{
   return lo + (float)(v - vMin) / (vMax - vMin) * (hi - lo);
}
//------
void loopReadInputs()
{
#ifdef ENABLE_IO
   // read and process moisture level:
   moistAna = limit(analogRead(8), 0, 1024);
   moistPctIst = scale(moistAna, moistAnaMin, moistAnaMax, 100.0, 0.0);
#endif
}
//-----
// helper to transfer FSM to other state:
void fsmGoTo(FsmState newState)
{
   stateCurrent = newState;
   WawiSrv.println((String)"New step = " + stateStrings[(int)stateCurrent]);
}
//-----
                  -----
// finite state machine:
void loopFsmStateLogic()
{
   do
   {
      // Move from one step to another if conditions are right:
      switch (stateCurrent)
      {
      case FsmState::STATE_0_VALVE_WATER_OFF:
          if (wateringEnable)
             fsmGoTo(FsmState::STATE 1 VALVE CLOSED);
          break;
      case FsmState::STATE 1 VALVE CLOSED:
          if (moistPctSoll > moistPctIst)
          {
             WawiSrv.wawiBreak(1, "Break when soil is to dry!");
             fsmGoTo(FsmState::STATE_2_VALVE_OPEN);
          }
          if (!wateringEnable)
             fsmGoTo(FsmState::STATE_0_VALVE_WATER_OFF);
          break;
      case FsmState::STATE_2_VALVE_OPEN:
          if (firstScanNewState)
          {
             WawiSrv.wawiBreak(2, "Break entering step 3.");
             timerFsm.setMs(valveTimeOpenMSec);
          }
```

```
if (timerFsm.isZero())
               fsmGoTo(FsmState::STATE_3_VALVE_CLOSED_HOLD);
           if (!wateringEnable)
               fsmGoTo(FsmState::STATE_0_VALVE_WATER_OFF);
           break;
       case FsmState::STATE_3_VALVE_CLOSED_HOLD:
           if (firstScanNewState)
           {
               WawiSrv.wawiBreak(3, "Break entering step 4.");
               timerFsm.setMs(valveTimeHoldMSec);
           }
           if (timerFsm.isZero())
               fsmGoTo(FsmState::STATE_0_VALVE_WATER_OFF);
           break;
       }
       firstScanNewState = (stateCurrent != statePrev);
       // report: translate stateCurrent into a string and put it in stateString:
       strcpy(stateString, stateStrings[(int)stateCurrent]);
       // display state change of the FSM in the output window:
       if (firstScanNewState)
       {
           statePrev = stateCurrent;
           WawiSrv.print("State change to: ");
           WawiSrv.println(stateString);
       }
   } while (firstScanNewState);
}
//-----
// Make alarm if dry is too long high:
void loopAlarming()
{
   // if watering is wateringEnabled, and moisture is too low a timer is started
   bool dry = moistPctIst < moistPctSoll;</pre>
   if (wateringEnable && !dry)
       dryAlarmTimer.setMs(dryAlarmTimer);
   // alarm after moistureTimeAlarmSec seconds of drought
   if (dryAlarmTimer.isZero())
       dryAlarm = HIGH;
   else
       dryAlarm = LOW;
}
//-----
                    _____
// finite state machine:
void loopFsmStateToOutputs()
{
   // Move from one step to another if conditions are right:
   switch (stateCurrent)
   {
   case FsmState::STATE_0_VALVE_WATER_OFF:
   case FsmState::STATE_1_VALVE_CLOSED:
   case FsmState::STATE_3_VALVE_CLOSED_HOLD:
       valveOpen = LOW;
       break;
   case FsmState::STATE_2_VALVE_OPEN:
       valveOpen = HIGH;
       break;
   }
}
```

```
11--
                        -----
void loopWriteOutputs()
{
   digitalWrite(VALVE IO, valveOpen);
   digitalWrite(DRY_ALARM_IO, dryAlarm);
}
void loop()
{
   loopReadInputs();
   loopFsmStateLogic();
   loopAlarming();
   loopFsmStateToOutputs();
   timerFsm.loop();
   totalValveTimer.loop(valveOpen);
   dryAlarmTimer.loop();
   WawiSrv.loop();
```

Fig 6.1. WawiWaterSensorValve.ino

```
#include <Arduino.h>
#define wawiVarTimer(X) wawiCompare(#X,(unsigned int) &X.m_t,sizeof(X.m_t),1);
class WawiTimer
{
private:
    unsigned long m_tPrev;
    bool m_activePrev;
public:
    WawiTimer(bool incremental = false);
    unsigned long long m_t;
    bool m_incremental;
public:
    bool isZero() const { return (m_t == 0); };
    void setMs(unsigned long long valueMs) { m_t = valueMs; m_tPrev = millis(); };
    void setMsAct(unsigned long valueMs) { m_tPrev = valueMs; };
    unsigned long long getValue() const { return m_t; };
    void loop(bool active = true);
    operator bool() const { return (m t == 0); };
};
```

```
Fig 6.2. WawiTimer.h
```

```
#include "WawiTimer.h"
#include "Arduino.h"
WawiTimer::WawiTimer(bool incremental)
{
    m_incremental = incremental;
    m_t = 0;
}
void WawiTimer::loop(bool active)
{
    unsigned long m, delta;
    if (!m_activePrev && active)
    {
        m_tPrev = millis();
        m_activePrev = active;
        return;
    }
    // calculate delta time between last loop and this loop;
```

```
if (m_activePrev)
{
   m = millis();
   delta = m - m_tPrev;
   m_tPrev = m;
}
// incrementing timer counter:
if (m_incremental && m_activePrev)
{
   m_t = m_t + delta;
   else
       m_t = 0xFFFFFFFFFFFFFF;
}
// decrementing timer counter:
if (!m_incremental && m_activePrev)
{
   if (m_t >= delta)
      m_t = m_t - delta;
   else
       m_t = 0;
}
m_activePrev = active;
```

Fig 6.3. WawiTimer.cpp

}