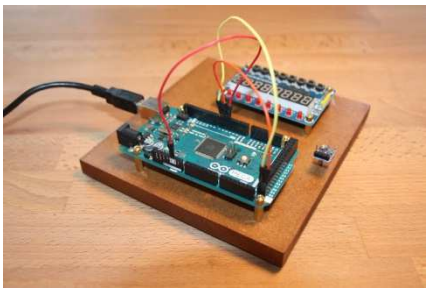
Recording variables and .print() output with WawiLib

Contents



1. Introduction

1.1. Objective of this document.

The first objective of this document is to describe how to use WawiLib to record variables. Points of interest are triggers to write data records to a disk and how to limit the sizes of the recorded datafiles. WawiRecUsb.ino, a demo sketch supplied with the WawiSerialUsb library, will be used to explain the concepts.

The second objective of this document is to describe how to use WawiLib to record .print() debug output to a PC disk file. WawiRecOutput.ino, a demo sketch supplied with the WawiSerialUsb library, will be used to explain the concepts.

1.2. Software and hardware requirements

The Arduino IDE (in this example 1.8.15) and WawiLib V2.0.x both need to be installed on your PC. The demo runs with licensed and unlicensed versions of WawiLib. During the grace period of 2 months, you can test and use all functions without registration. After this period registration is required in order to access all functions. At this time registration is free. In the future a small contribution might be required to register in order to support the website.

You also need a program to open the recorded data files: Excel or OpenOffice to open .xlsx files, a text editor to open .csv files and an XML viewer (I use XML Notepad from Microsoft) to open .xml files.

In this demo, the USB programming port of the Arduino is used as the communication interface between WawiLib and your Arduino shield. This demo can easily be converted to other types of communication links. The WawiLib getting started demos for serial, Ethernet and Wi-Fi communication can be used as a base for the conversion to another interface type. The only thing you have to do is replace the initialization code as in the demo's for WawiWifi and WawiEthernet.

The hardware you need is an Arduino board, a USB programming cable, 3 Dupont male-male (breadboard) wires and a Windows PC (32 or 64 bit). In this demo, we will use the Arduino UNO board but other boards can be used in a similar or even identical way.

The data recorder that is part of WawiLib is typically used to record all kinds of I/O signals. In this demo, we will record mainly internal variables of the Arduino. This choice was made to keep things simple and easy to understand. On www.sylvestersolutions.com, you will find application notes that enable you to build other applications using specific hardware.

1.3. Required user experience

You should have completed the tutorial "Getting started with WawiLib USB" and "Debugging with WawiLib USB". There are no specific additional requirements.

2. The “WawiRecUSB” Demo sketch example

2.1. Concept of “WawiBlinkRecUSB”

This application builds further on the sketch “WawiBlinkDebugUsb” from the demo “Debugging with WawiLib”. WawiBlinkRecUsb is in fact WawiBlinkDebugUsb with its errors corrected.

The objective of the program is to blink 5 sec after a pulse on digital input 5, to blink 7 sec after a pulse on input 6 and to blink 10 seconds after a pulse on input 7. If one of the IO’s remains high, blinking continues.

The program prints diagnostic output to the WawiLib output window when the LED is blinking:

- *Counting down:*
- The value of *blinkTimeActual*;
- *LED is ON.*
- *LED is OFF*"

2.2. Download and execute “WawiBlinkRecUSB”

✓ Open the demo sketch using the menu “File\Examples\WawiSerialUsb\WawiBlinkRecUSB” in the Arduino IDE.
✓ Connect inputs 5, 6 and 7 to the GND pins of your board.
✓ Compile and download WawiBlinkRecUSB to your Arduino board.

```
/*
* Project Name: WawiRecUsb
* File: WawiRecUsb.ino
*
* Detailed manual:
* www.SylvesterSolutions.com\documentation -> "Recording variables with WawiLib.pdf"
*
* Description: demo file library for WawiSerialUsb library.
* Data recorder demo.
* => Record values of variables to disk
* => Record .print() output to disk
* Use the USB programming port to make connection with the Arduino board.
* Variables can be checked & modified with the WawiLib-PC software.
*
* Author: John Gijs.
* Created March 2020
* More info: www.sylvestersolutions.com
* Technical support: support@sylvestersolutions.com
* Additional info: info@sylvestersolutions.com
*/

#include <WawiSerialUsb.h>

WawiSerialUsb WawiSrv;
#define LED 13 // blinking light
#define IN_5 5 // light start blinking switch 1
#define IN_6 6 // light start blinking switch 2
#define IN_7 7 // light start blinking switch 3

// variables for demo:
long int blinkTimeActual = 0; // counter blink active (milliseconds)
long int blinkTimeTarget[] = { 5000, 7000, 10000 }; // bug 1: should be { ..., ...,
10000};
```

```
bool digInput5; // state of digital input 5
bool digInput6; // state of digital input 6
bool digInput7; // state of digital input 7
bool led; // state of led
int loopCounter;

// make variables of interest know to WawiLib:
void wawiVarDef()
{
    WawiSrv.wawiVar(digInput5);
    WawiSrv.wawiVar(digInput6);
    WawiSrv.wawiVar(digInput7);
    WawiSrv.wawiVar(led);
    WawiSrv.wawiVar(blinkTimeActual);
    WawiSrv.wawiVar(loopCounter);
    WawiSrv.wawiVarArray(blinkTimeTarget);
}

void setup()
{
    Serial.begin(115200);
    // initialize WawiLib library:
    WawiSrv.begin(wawiVarDef, Serial, "MyArduino");
    pinMode(LED, OUTPUT);
    pinMode(IN_5, INPUT);
    pinMode(IN_6, INPUT);
    pinMode(IN_7, INPUT);

    WawiSrv.wawiBreakDisable();
}

void loop()
{
    digInput5 = digitalRead(IN_5);
    digInput6 = digitalRead(IN_6);
    digInput7 = digitalRead(IN_7); // bug 2: should be digInput7 = ...


    if (digInput5)
        blinkTimeActual = blinkTimeTarget[0]; // bug 3: should be activeMsSetpoint[0]

    if (digInput6)
        blinkTimeActual = blinkTimeTarget[1];

    if (digInput7)
        blinkTimeActual = blinkTimeTarget[2];

    if (digInput5 || digInput6 || digInput7)
    {
        WawiSrv.wawiBreak(1, "breakpoint after write to activeMsCounter hit");
    }

    while (blinkTimeActual > 0) // bug 4: should be activeMsCounter > 0
    {
        WawiSrv.wawiBreak(2, "In while loop");

        WawiSrv.print("Counting down:");
        WawiSrv.println(blinkTimeActual);

        WawiSrv.println("LED is ON.");
        led = HIGH;
        digitalWrite(LED, led);
```

```
        WawiSrv.delay(500);
        blinkTimeActual = blinkTimeActual - 500;

        WawiSrv.println("LED is OFF.");
        led = LOW;
        digitalWrite(LED, led);
        WawiSrv.delay(500);
        blinkTimeActual = blinkTimeActual - 500;
    }
    WawiSrv.loop();
    loopCounter++;
}
```

Fig. 2.1. WawiRec source code.

The demo sketch contains the variables *delayOn* and *delayOff.* They determine the blinking timing of the LED. The variables *activeMsCounter* and *activeMsSetpoint* determine the time interval the LED will remain blinking. *DigInput5*, *digInput6* and *digInput7* mirror the state of the 3 digital inputs 5, 6 and 7. The function wawiVarDef shares the addresses of the variables with WawiLib.

If you have studied the WawiDebugUSB demo, you will be familiar with the rest of the program: *WawiServ.begin(wawiVarDef,Serial,"MyArduino")* initializes the library for communication over the USB port. *wawiVarDef()* is the function where addresses of the variables are shared. "MyArduino" is used as name reference for the board.

3. WawiLib data recording to disk file

3.1. Introduction

One of the properties of the Arduino environment is that it requires a bit more than basic knowledge to record data to your PC.

Programming data communication via serial, Ethernet, Wi-Fi and USB interfaces is not so easy within Windows. For sure if you want to do this the right way (overlapped I/O, non-blocking, multithreaded, automatic re-connect etc.).

Originally, WawiLib did not contain any data recording functions. As I wanted to create a PID controller and record the signals, I decided to add recording functions to the program. Typical applications where recording can be used are physics experiments at school or elsewhere.

Suppose you want to do some thermal experiments using a calorimeter. You can of course write down on paper the temperature of the calorimeter in time. But automated data recording creates additional value in this kind of experiment: increased accuracy, technical challenge and the notion of “big data”. Not to forget that writing temperature values down on paper every 10 seconds is boring and outdated.

Connecting a DS18B20 high resolution temperature sensor to an Arduino board is not so challenging. But getting the measured data in an Excel table for further processing requires more specialized knowledge. With the data recording functions integrated in WawiLib, everybody will be able to create data recording files by configuring the right settings within WawiLib on the PC.

3.2. Data recorders

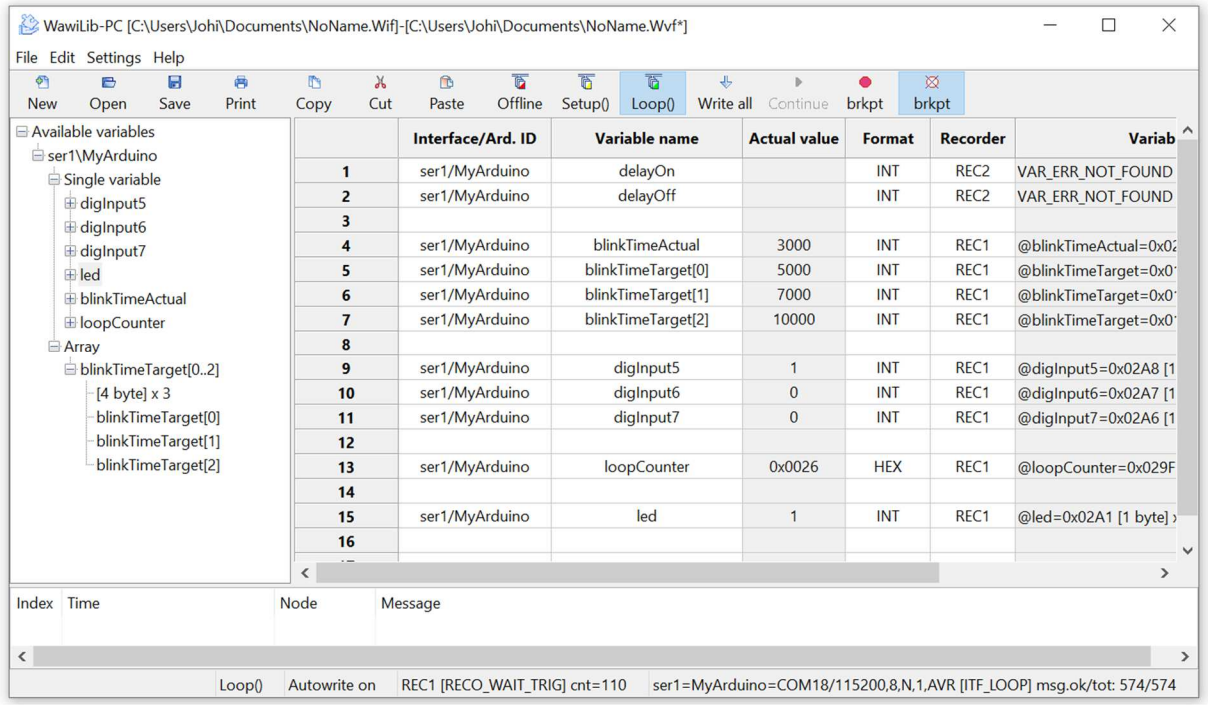


Fig. 3.1. Variables with multiple data recorders.

Data recording in WawiLib is implemented with a data recorder object. In the figure above, you see 11 variables. Data recorder REC1 will record the first 2 variables. Data recorder REC2 will record the last 9 variables. Many data recorders can work next to each other, each of them with their own file type and recording settings.

Every time a data recorder does a recording, it takes the actual value of all its variables from the PC memory and writes the values to a disk file or memory file in a single write job. Each recorder has its own data file. At the beginning of the line, the timestamp indicating the current PC time is added (details: see later).

The value of these variables in the PC will be more or less accurate depending on the speed used to refresh them. Refreshing is done by reading the values from the Arduino memory. There is a general refresh setting that can be overridden individually for each variable. To change general settings:

✓ In WawiLib, go to the menu “Settings/User preferences and license”.

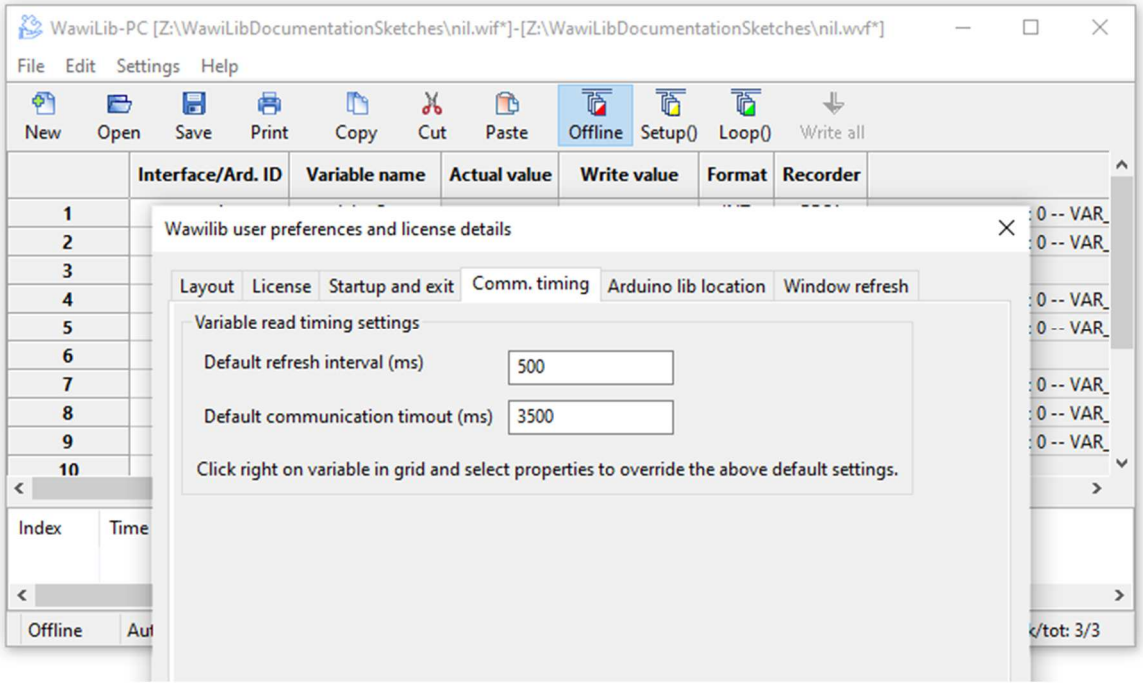


Fig. 3.2. Modify the default refresh interval of variable Arduino ⇔ WawiLib memory.

In the fig. 3.2., you see the default settings for the refresh interval of the variables read from the Arduino. If you make this value too small, the communication link with your Arduino will be overloaded. If you make it too large, there will be too much lag between the values recorded and the real values of the variables.

✓ Enter the value of 250 ms for the default refresh interval.
✓ Press “ok”.

You can override the general timing for each individual variable (fig. 3.3):
✓ Click right on a variable and select “variable properties”.

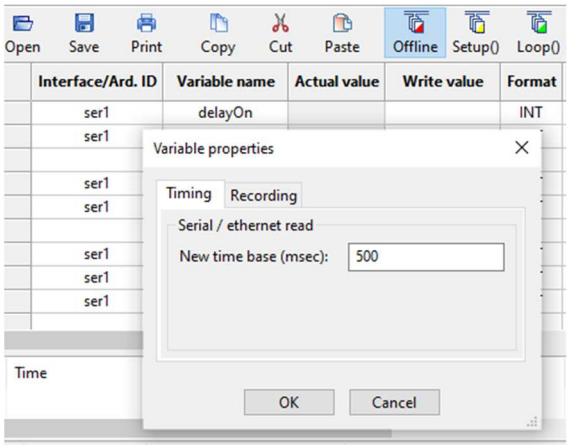


Fig. 3.3. Modify the individual refresh interval of variable Arduino ⇔ WawiLib memory.

WawiLib can create 3 types of data recording files:

- xml: extensible Markup Language files
- xlsx: Microsoft Excel/Open Office compatible files
- csv: comma separated value files

Another topic is what to do with the recorded data when WawiLib is set offline and then reconnected by the user. Some users require that the existing data file is overwritten, others expect the new data to be appended to the data file and yet another might require a new file to be started.

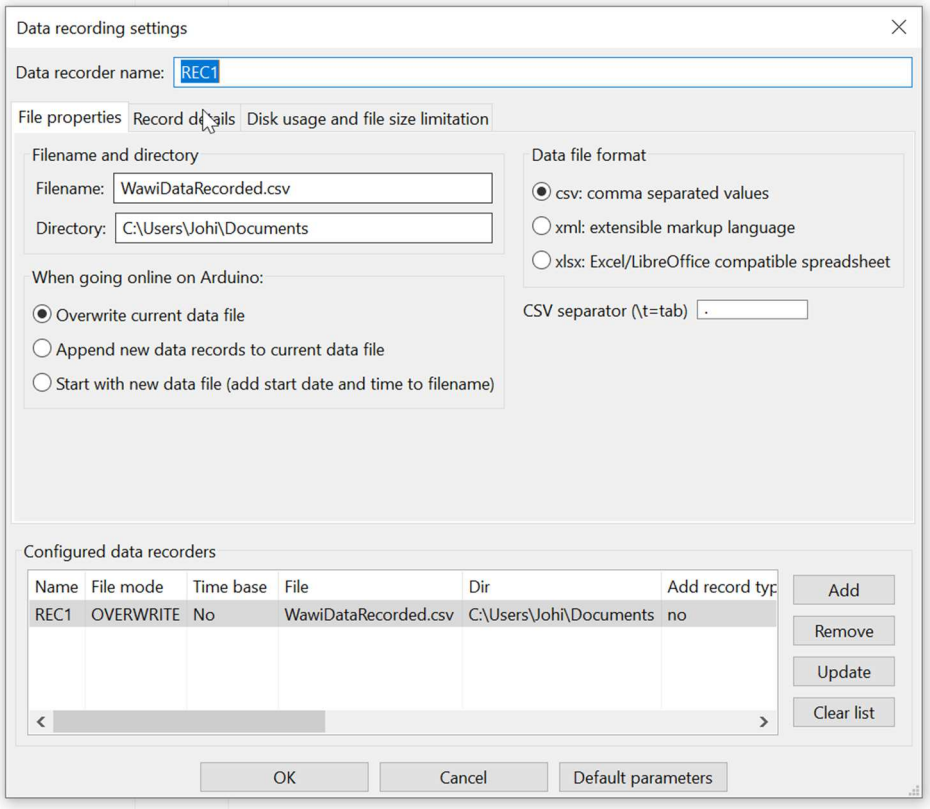


Fig. 3.4. Data recorder settings with various options for file types, and file approach (append overwrite…).

All 3 options are available within WawiLib. If you choose to create a new file, the date and the time the file was created will be added to the data file name to distinguish one file from another. The name can be in UTC or local time. UTC is not impacted by winter time – summer time changes but local time depends on the area where you live.

3.3. Data recording trigger aspects

Different applications have different data trigger requirements. The calorimeter application from the previous section for example could require data recording every 10 seconds (time-based recording).

Another type of application would be registration of movement detection with a PIR infrared sensor module. The PIR output would be connected to an Arduino input. In this application, time-based registration is not the right way to go. This type of application requires an additional (time stamped) data record when the value of the output of the detector has changed. We are not interested in recording the PIR signal unless the output of the sensor changes (on-change recording).

- ✓ In WawiLib, go to “Settings/Data Recording”.
- ✓ Select the tab “Record details”.

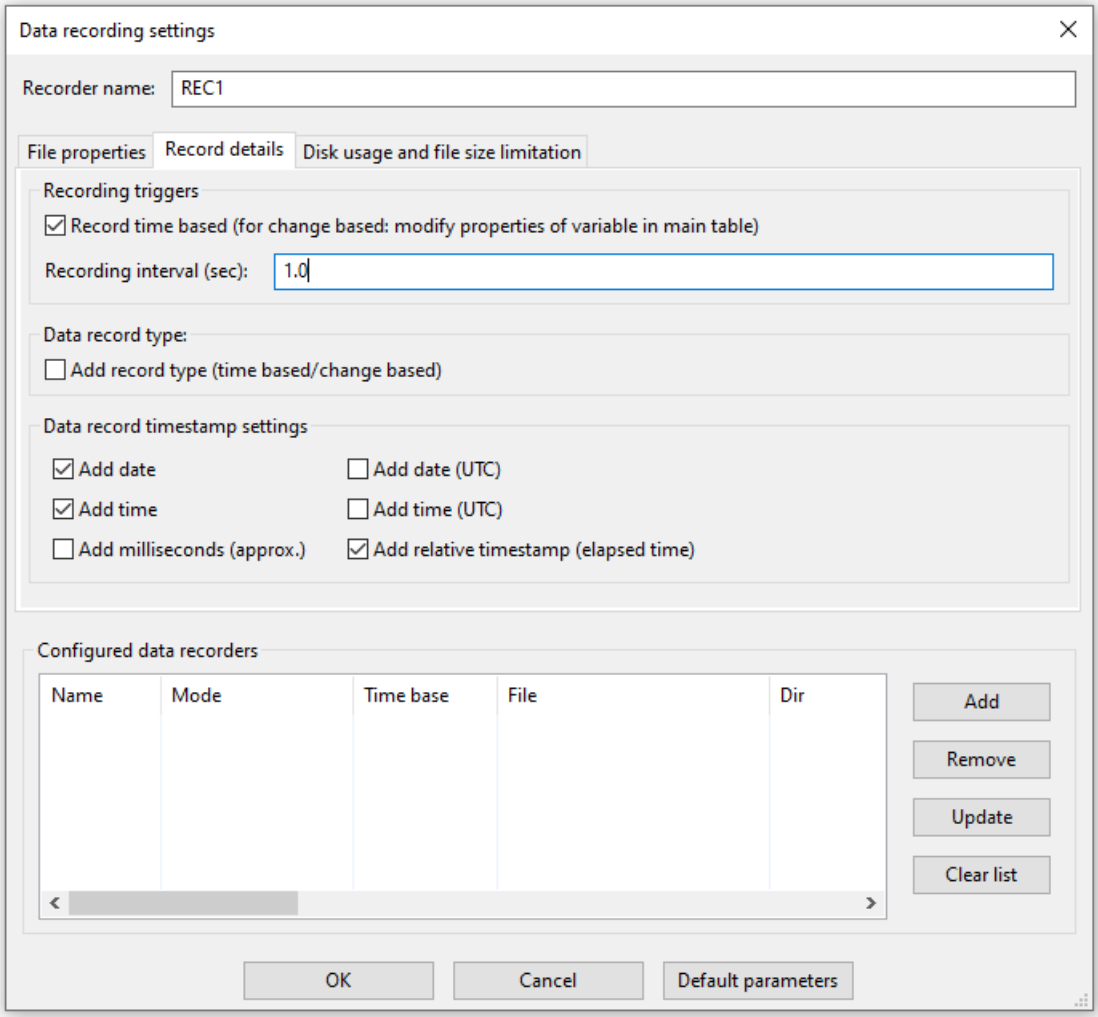


Fig. 3.5. Modify the recording interval of a data recorder (WawiLib memory ⇔ Disk file).

If you check “Record time based”, the data recorder will store all variable values each time the recording time interval has expired. The recording interval is specified in the “Recording interval (sec)” field. If you want to use on-change recording, you need to uncheck “Recording time based”.

While using on-change recording, the variables will still be polled in the Arduino memory according to the communication timing settings. But the moment a variable change is detected in the PC, a trigger is sent to the data recorder to write the value of all the variables associated with this data recorder immediately to disk (or to memory for .xlsx files). Whether a variable change will trigger a write of all variables associated with the recorder or not can be defined in the properties section of each individual variable. You can access the properties of a variable by clicking right on the variable in the grid of the main window.

Typically you can use an integer that increments on each event, whenever its value increases, WawiLib sees this and writes all variables related to the current data recorder to dis.

The recorder does not wait for its recording time interval to elapse to trigger a write job. When the internal recorder timer elapses, data are written again to disk (if enabled). This function is very handy to store events when and if they happen.

The trigger to write data “on change” mentioned in the previous paragraph does not even have to be part of the variables written to a disk. It can act as a trigger that is not part of the recorded data. This function can also be configured in the properties section of the variable.

Each data record can be accompanied by a time stamp. The time stamp can contain the current date and time in local time or UTC (universal time coordinated) or both time stamps. An approximate time stamp in milliseconds and the number of seconds since the start of the data recording can also be added.

3.4. Data recording detailed example

3.6.1. Recorded variables

✓ Fill in the WawiLib table as in fig. 3.6. and press ‘Setup()’.

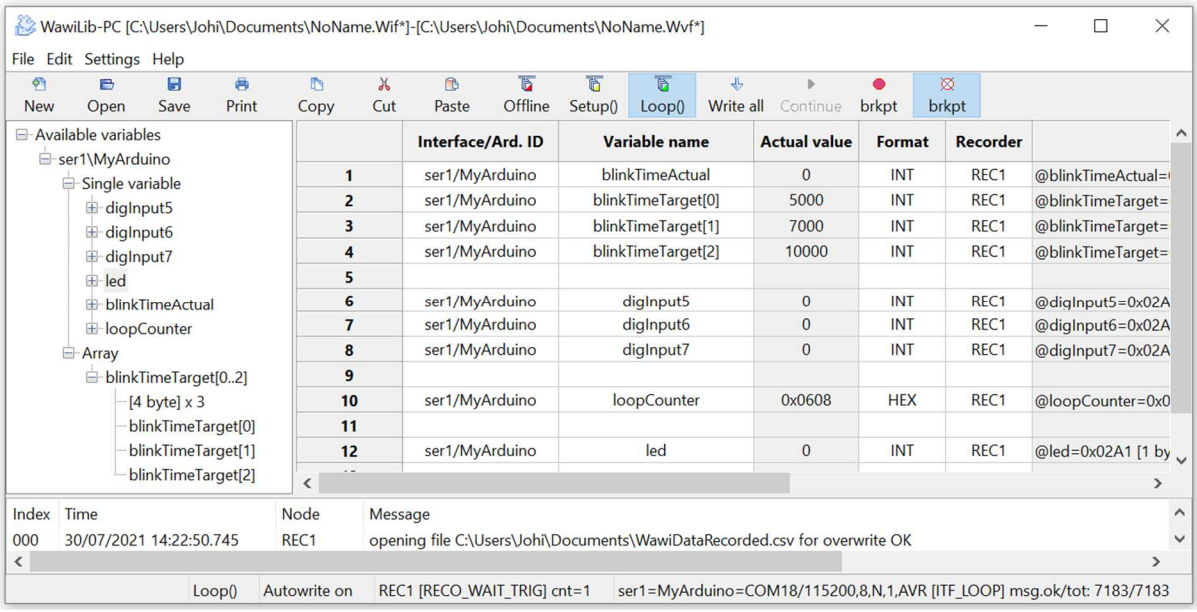


Fig. 3.6. Variables linked to recorder 1.

- ✓ Disable and enable output window settings as in fig 3.6.
- ✓ (Enable “Display data recording”. => What is written to disk during variable recording will be shown in the output window.)
- ✓ (Enable “Display output window recording”. => What is written to disk during output recording will be shown in the output window.)
- ✓ (Enable “Automatic scroll” => The window will scroll to the last message added.)

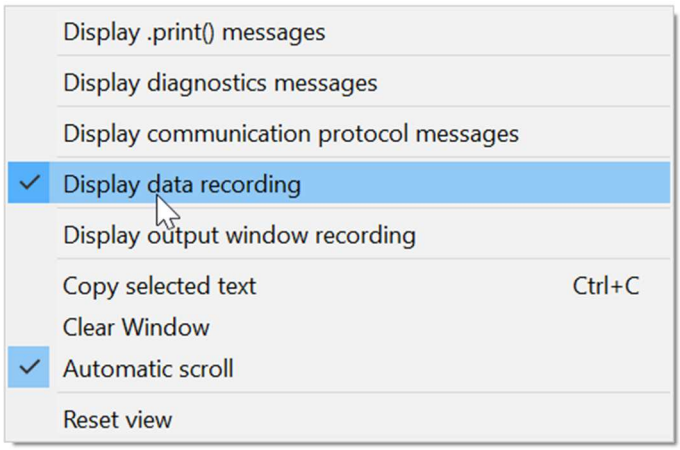


Fig. 3.6. Enable display data recording in the output window & automatic scroll.

3.6.2. Time based recording of a variable or a series of variables

- ✓ Go to “Settings/Data Recording…”

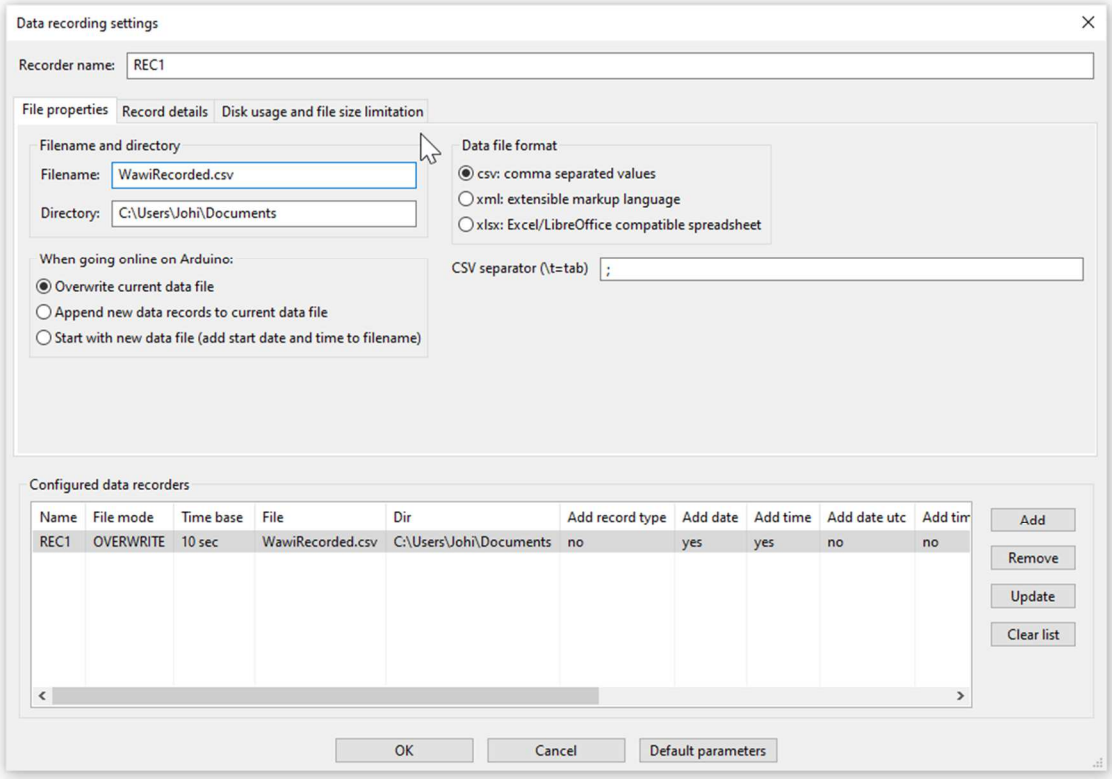


Fig. 3.7. Define REC1 as a data recorder time based.

- ✓ Fill in the table as indicated in fig. 3.7.

- ✓ Press “Add” and “Ok”.
- ✓ Select tab 2.
- ✓ Select REC 1 in the list control.
- ✓ Change recording interval to 1.0 sec
- ✓ Press “Update”

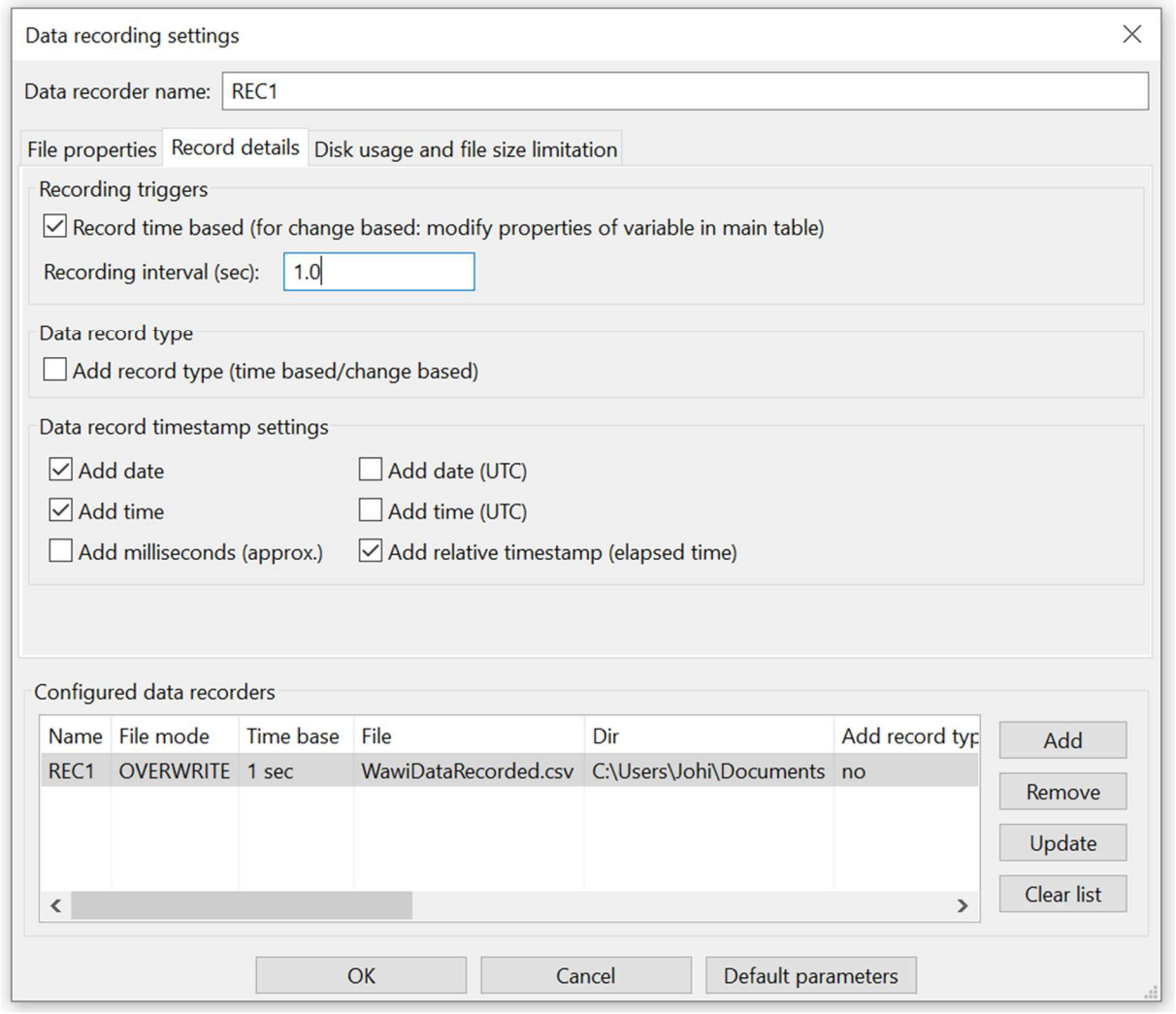
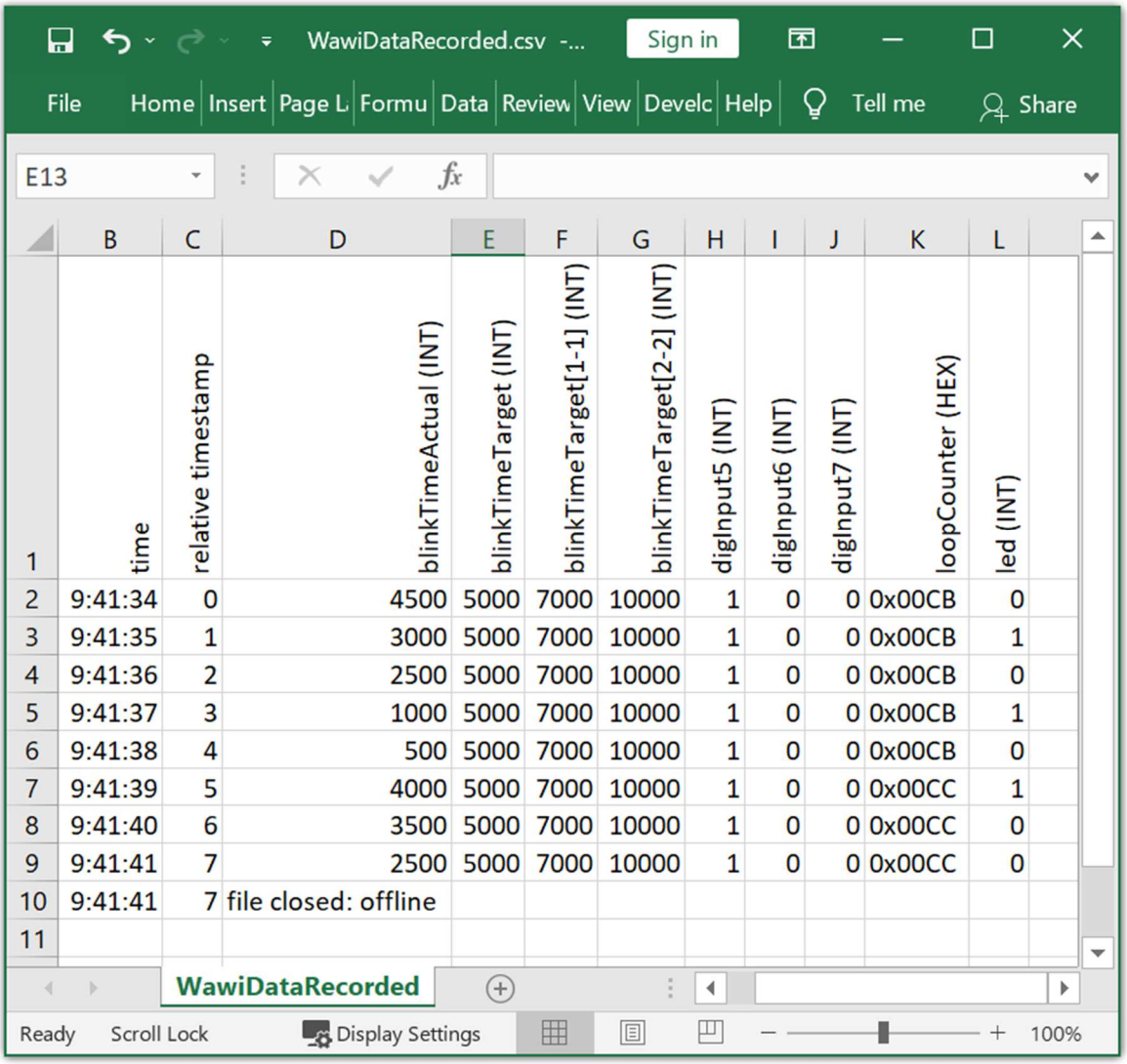


Fig. 3.8. Change time base of REC1 from 10 sec to 1 sec.

- ✓ Press OK.
- ✓ Press ‘Setup ()’.
- ✓ Press “Offline”.

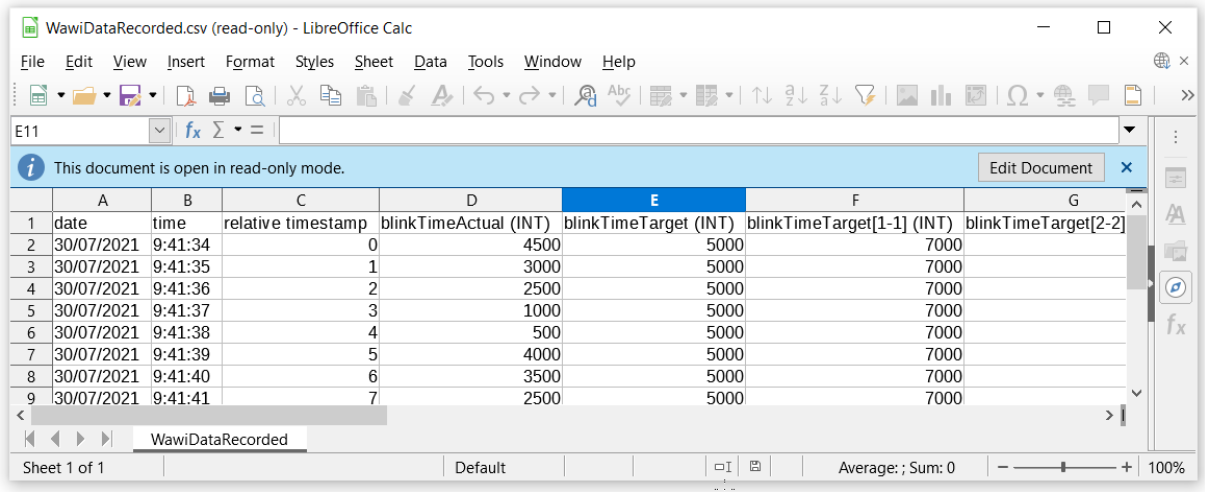
In the output window, you see at line 003 the first record written to the data file. This is the title of the various columns (including the variable names). In the output window, you see what is written by Rec 1: delayOn and delayOff are both 500. On line 014 in the output window, you see the footer that is written when WawiLib goes offline.

- ✓ Start Excel or LibreOffice calc.
- ✓ Drag and drop the file (file name and location as indicated as indicated on line 014 above) in the Excel grid:



	B	C	D	E	F	G	H	I	J	K	L
1	time	relative timestamp	blinkTimeActual (INT)	blinkTimeTarget (INT)	blinkTimeTarget[1-1] (INT)	blinkTimeTarget[2-2] (INT)	digInput5 (INT)	digInput6 (INT)	digInput7 (INT)	loopCounter (HEX)	led (INT)
2	9:41:34	0	4500	5000	7000	10000	1	0	0	0x00CB	0
3	9:41:35	1	3000	5000	7000	10000	1	0	0	0x00CB	1
4	9:41:36	2	2500	5000	7000	10000	1	0	0	0x00CB	0
5	9:41:37	3	1000	5000	7000	10000	1	0	0	0x00CB	1
6	9:41:38	4	500	5000	7000	10000	1	0	0	0x00CB	0
7	9:41:39	5	4000	5000	7000	10000	1	0	0	0x00CC	1
8	9:41:40	6	3500	5000	7000	10000	1	0	0	0x00CC	0
9	9:41:41	7	2500	5000	7000	10000	1	0	0	0x00CC	0
10	9:41:41	7	file closed: offline								
11											

Fig. 3.9. Data recorded opened in Microsoft Excel.



	A	B	C	D	E	F	G
1	date	time	relative timestamp	blinkTimeActual (INT)	blinkTimeTarget (INT)	blinkTimeTarget[1-1] (INT)	blinkTimeTarget[2-2]
2	30/07/2021	9:41:34	0	4500	5000	7000	
3	30/07/2021	9:41:35	1	3000	5000	7000	
4	30/07/2021	9:41:36	2	2500	5000	7000	
5	30/07/2021	9:41:37	3	1000	5000	7000	
6	30/07/2021	9:41:38	4	500	5000	7000	
7	30/07/2021	9:41:39	5	4000	5000	7000	
8	30/07/2021	9:41:40	6	3500	5000	7000	
9	30/07/2021	9:41:41	7	2500	5000	7000	

Fig. 3.10. Data recorded opened in LibreOffice Calc.

- ⇨ If excel does not process the .csv file properly y can use the command Data/From Text/CSV to import the .csv formatted data.
- ⇨ Above you see the reslult of opening the file in LibreOffice Calc.

3.6.3. Change based recording of a variable or a series of variables

✓ Go to “Settings/Data Recording…”.
✓ Press “Clear List”.
✓ Fill in the fields as indicated in fig. 3.11:

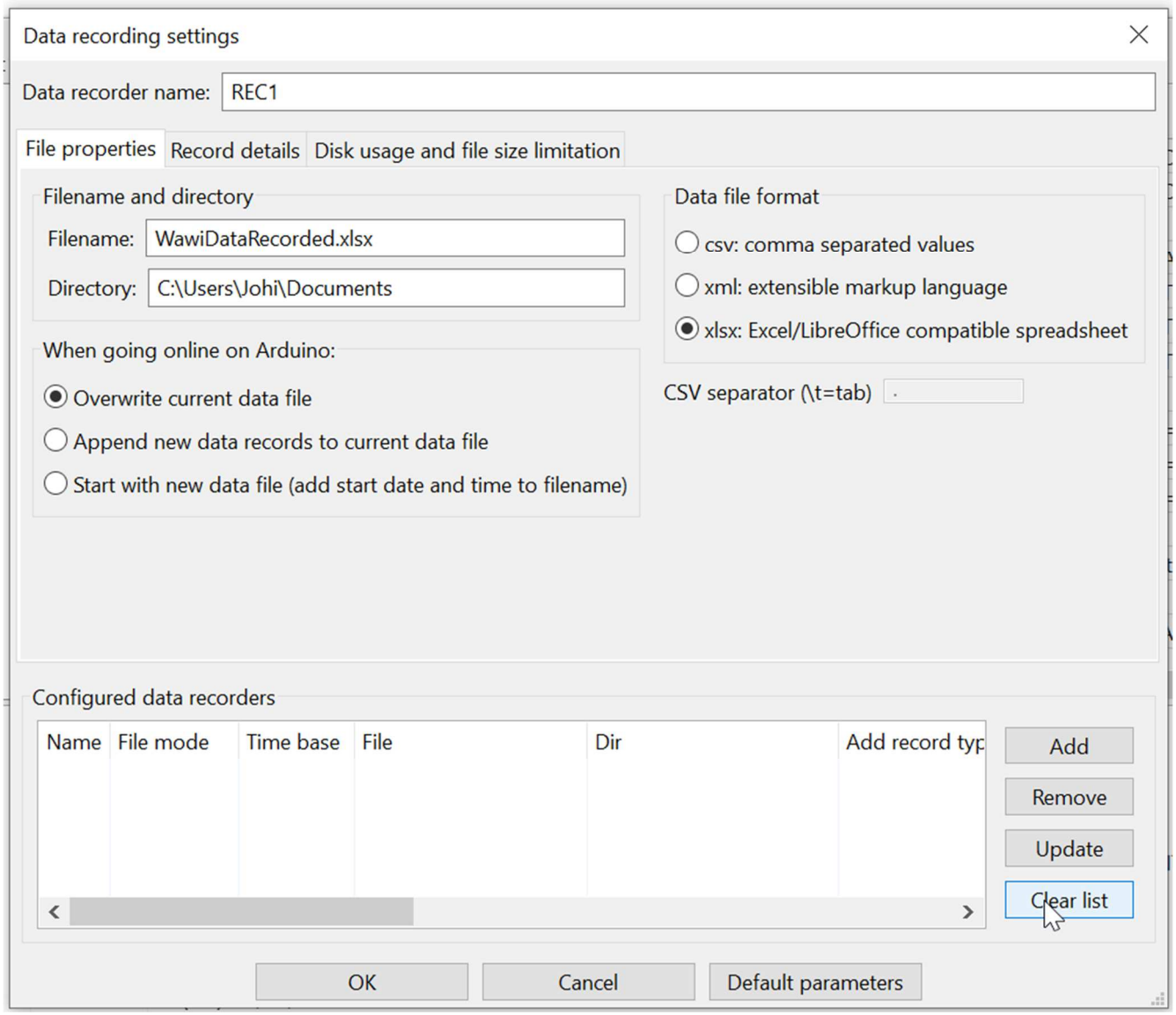


Fig. 3.11. Creating a new data recorder, XLSX file format and overwrite mode.

✓ Select the tab “Record details” (fig. 3.12.).
✓ Unselect (disable) “Recording time based”.

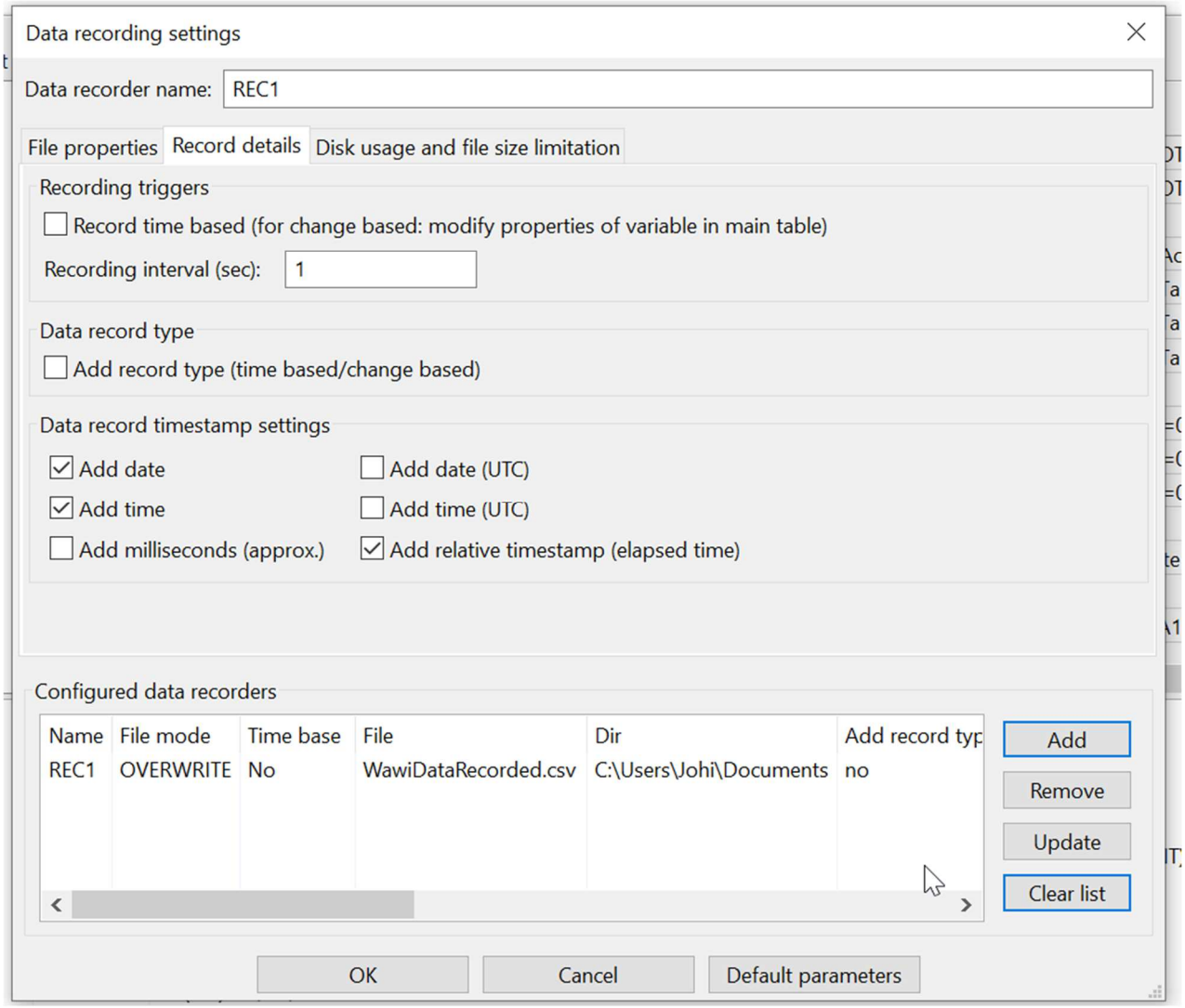


Fig. 3.12. Creating a new data recorder, disable time interval based recording.

- ✓ Press “ADD”.
- ✓ Press “OK” (close the dialog box).
- ✓ In the main variable grid, change the name of the recorders to REC2 as indicated below.
- ✓ In the main variable grid, select line 7 with the variable “*blinkTimeActual*”. (fig. 3.13.)
- ✓ Right-click and open the menu “Variable properties”.
- ✓ Go to Tab “Recording” in the dialog box.
- ✓ Check “Recording enabled” and “Change of variable triggers data recorder”.

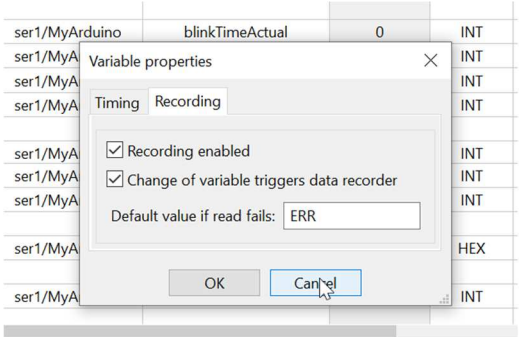


Fig. 3.13. Make *blinkTimeActual* a trigger for variable change recording.

⇨ Note: If you disable (fig 3.13.) Recording enabled, this variable will trigger recording of the complete set of variables in REC1 but blinkActual will not be part of the recorded data.

✓ Enable “Trace data recording “ in the output window.
✓ Press “OK” and Press ‘Setup()’.
✓ On your Arduino board, connect Input 5 to 5V using the breadboard wire.
✓ Connect the input 5 6 and 7 to GND.
✓ Connect the input 5 to 5V.
✓ Connect the input 5 to GND.
✓ Look at the output window.
✓ Wait 10 seconds.
✓ Press “Offline”.

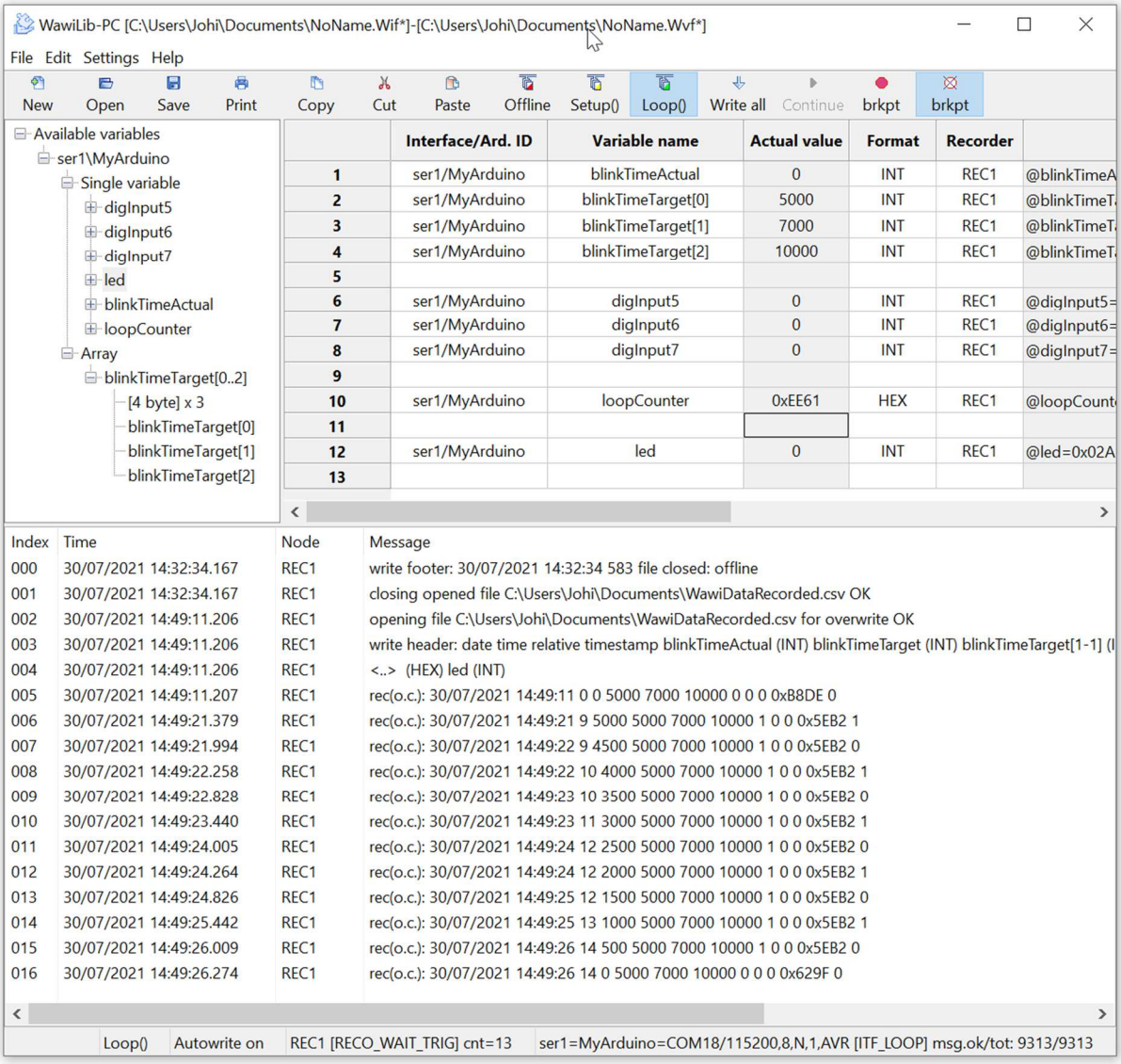
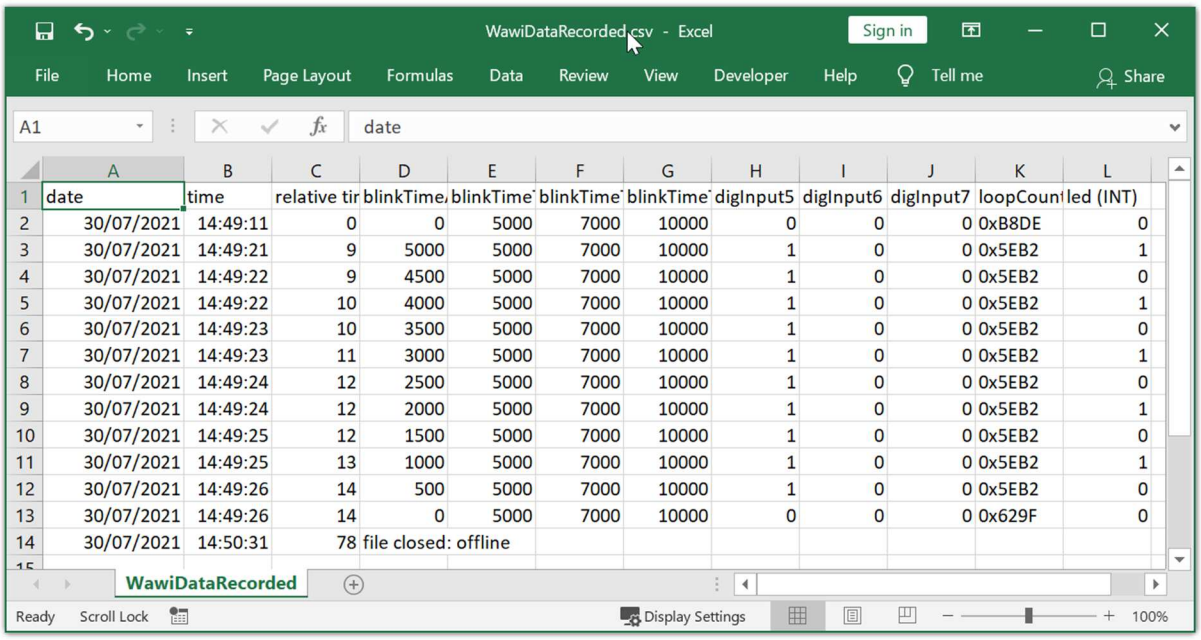


Fig. 3.13. On change recording example result.

On each alternation of *blinkTimeActual*, a new data set will be written to the data file. Each write operation is also visible in the output window. Once *blinkTimeActual* no longer changes, the recording stops.

✓ Open the file (see line 002 of the output window) in Excel or LibreOffice calc:



	A	B	C	D	E	F	G	H	I	J	K	L
1	date	time	relative tir	blinkTime	blinkTime	blinkTime	blinkTime	digInput5	digInput6	digInput7	loopCount	led (INT)
2	30/07/2021	14:49:11	0	0	5000	7000	10000	0	0	0	0xB8DE	0
3	30/07/2021	14:49:21	9	5000	5000	7000	10000	1	0	0	0x5EB2	1
4	30/07/2021	14:49:22	9	4500	5000	7000	10000	1	0	0	0x5EB2	0
5	30/07/2021	14:49:22	10	4000	5000	7000	10000	1	0	0	0x5EB2	1
6	30/07/2021	14:49:23	10	3500	5000	7000	10000	1	0	0	0x5EB2	0
7	30/07/2021	14:49:23	11	3000	5000	7000	10000	1	0	0	0x5EB2	1
8	30/07/2021	14:49:24	12	2500	5000	7000	10000	1	0	0	0x5EB2	0
9	30/07/2021	14:49:24	12	2000	5000	7000	10000	1	0	0	0x5EB2	1
10	30/07/2021	14:49:25	12	1500	5000	7000	10000	1	0	0	0x5EB2	0
11	30/07/2021	14:49:25	13	1000	5000	7000	10000	1	0	0	0x5EB2	1
12	30/07/2021	14:49:26	14	500	5000	7000	10000	1	0	0	0x5EB2	0
13	30/07/2021	14:49:26	14	0	5000	7000	10000	0	0	0	0x629F	0
14	30/07/2021	14:50:31	78	file closed: offline								

Fig. 3.14. On change recording example result.

In the table above, you can see that the status changes of *blinkTimeAcutal* are the only ones to trigger the addition of a new data record to the data recording file. Other variables are recorded at the same moment. As *digInput5* goes to 1, *blinkTimeAcutal* starts decreasing. Do note that digInput5 remains recorded as 1 because it is not updated as long as the CPU is executing the while() loop.

If we would enable “Recording time based” for recorder REC2, it would record data both time based and change based. Typically, you could decide to record at a slow pace time based and use “on change” to have updated data when a trigger comes indicating something interesting happened.

3.5. Data recorder storage aspects

One of the biggest challenges of data recording is how to manage your disk space. Imagine a weather station application. It runs day in and day out, recording its data in a single file. This would be not so handy because the file risks to become so large that it cannot be handled any more.

In the same sense there are limits to the amount of space a program can occupy on your hard disk. You do not want to get into trouble with other programs because WawiLib recorded data are eating up too much of your free disk space.

In order to tackle these 2 issues, every data recorder in WawiLib is able to delete its old data files if the amount of disk space used by the data recorder is too large.

Old data files can only be deleted once they are closed. The disk cleanup is triggered each time the data recorder closes a data file. So you need to restart with a new file regularly.

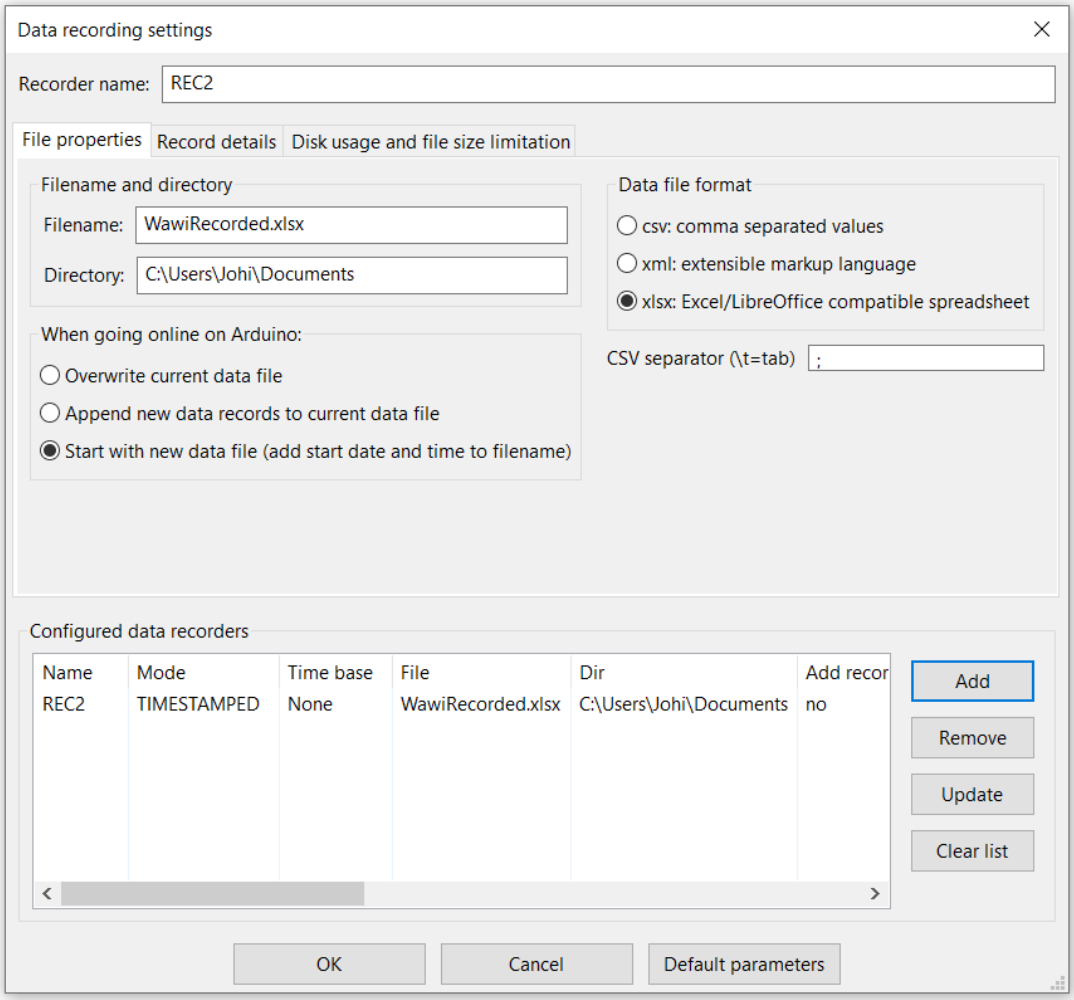


Fig. 3.15. Data recorder that creates a new data file each 15 minutes, hour, or day.

In the data recorder settings on tab 3 (“Disk usage and file size limitation”) you can enable the function to limit the amount of disk space used by the current data recorder. Files that have the same name as indicated in the “Filename tab” will be deleted, the oldest one first. This option is only valid for recorders that create new files every 15 minutes, every hour or every day.

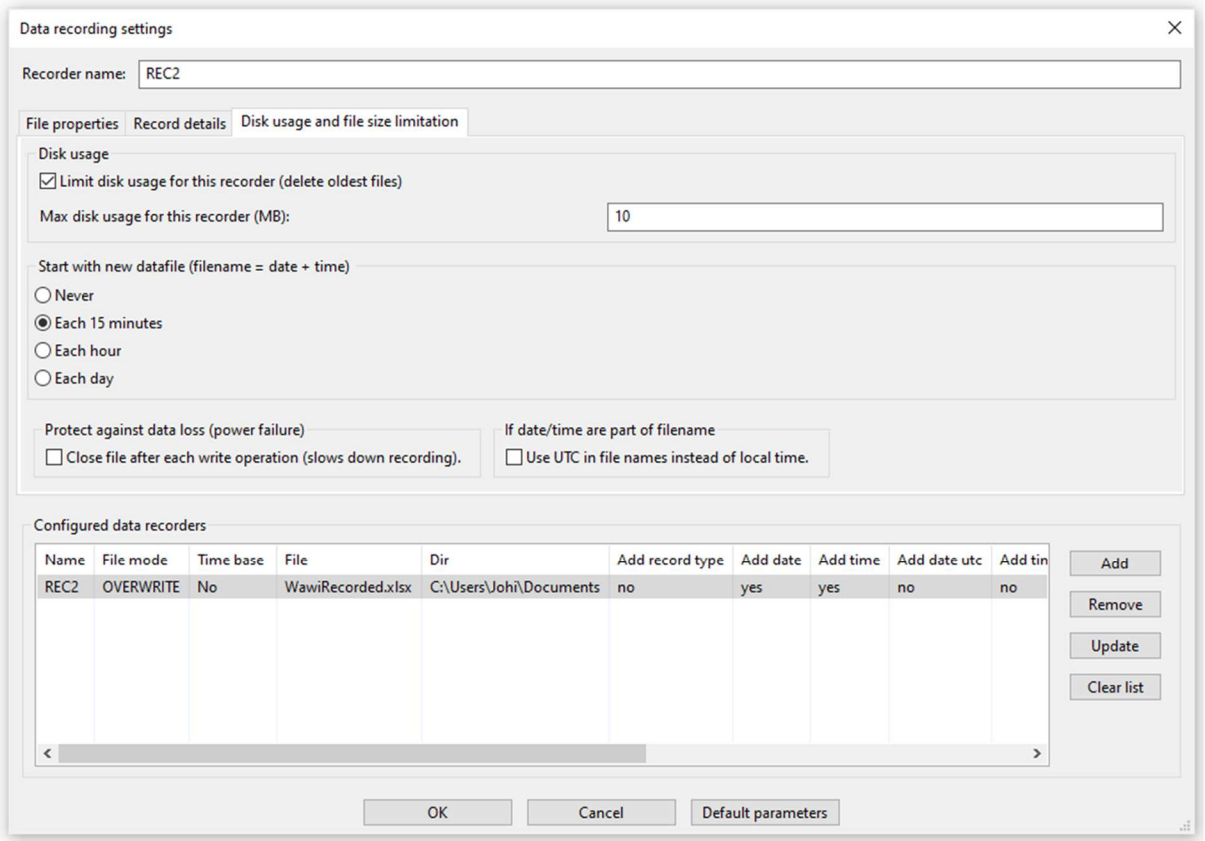


Fig. 3.16. Data recorder that creates a new data file each 15 minutes.

Above you see the check box “Close file after each write operation. This option makes sure your data ends up on disk immediately. In case of the weather station application, recording data each 15 minutes is more than fast enough. If we close the file after each write, we are much less vulnerable to failures of the grid feeding our computer and other anomalies.

4. WawiLib .print() recording to disk file.

4.1. Introduction

One of the properties of the Arduino environment is that it requires a bit more than basic knowledge to record data to your PC.

Programming data communication via serial, Ethernet, Wi-Fi and USB interfaces is not so easy within Windows. For sure if you want to do this the right way (overlapped I/O, non-blocking, multithreaded, automatic re-connect etc.).

In many applications one needs the ability to register output of a program during a prolonged period of time. Typically this can be an application that contains a very difficult to find bug or this can be an application where the user needs to register events that happen from time to time.

Originally, WawiLib did not contain any output recording functions. From time to time Sylvester Solutions does provide consultancy services and in one case the user wanted to register continuously the output of the sketch on a remote server via Wi-Fi. This is where the idea to create output recorders very similar to the data recorder in the previous chapters originated.

4.2. Define an output recorder

- ✓ Go to “Settings/Data Recording…”.
- ✓ Press “Clear List”.

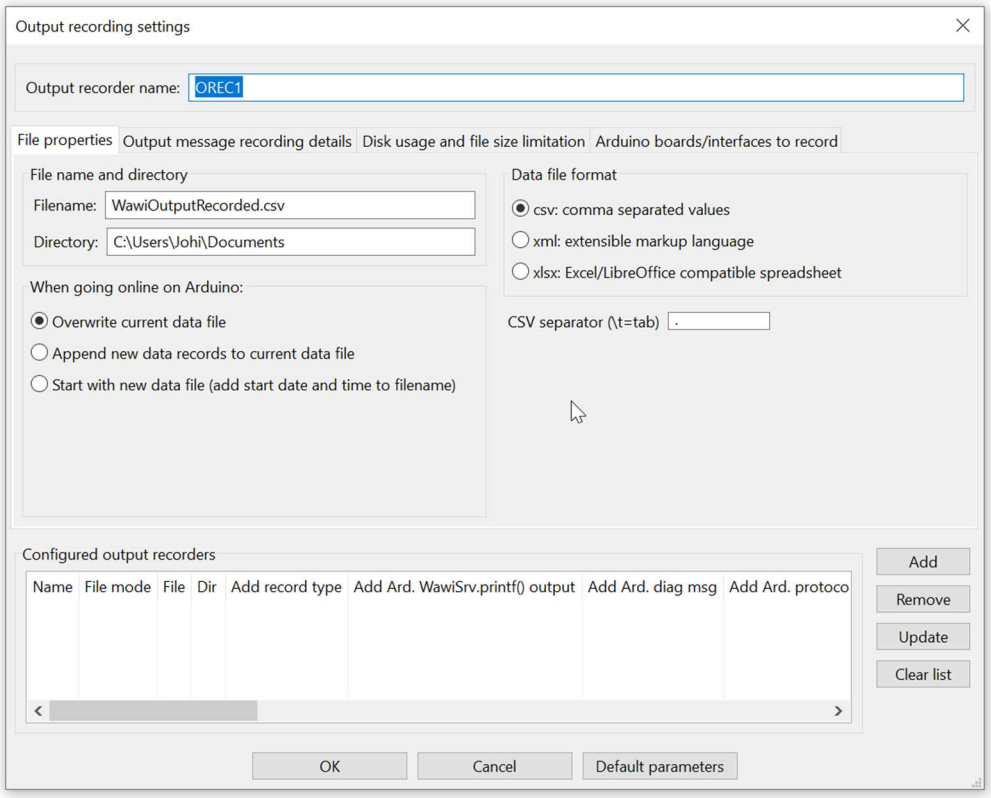


Fig 4.1. define an output recorder (file type)

- ✓ Fill in the fields as indicated in fig. 4.1

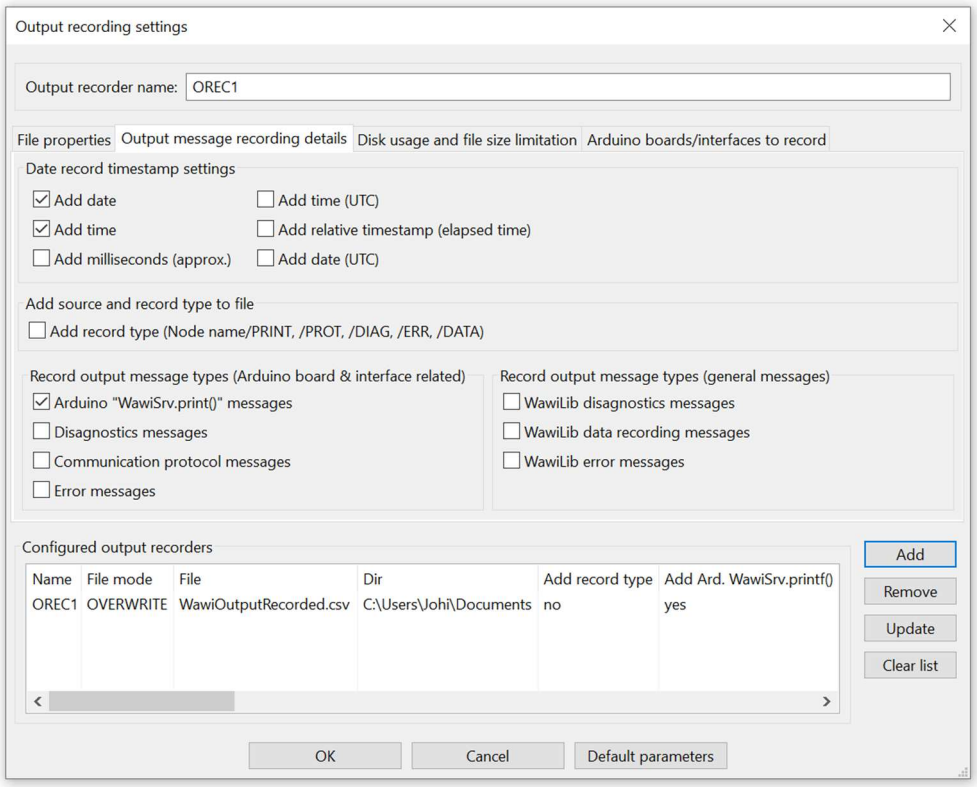


Fig 4.2. select the data to be recorded with the output recorder.

- ✓ Go to tap 2
- ✓ Fill in the fields as in fig. 4.2.
- ✓ Press “Add”
- ✓ Press “OK”
- ✓ Press “Setup” in the main tool bar.
- ✓ Enable the output window Diagnostic message display as in Fig. 4.3.

- ✓ Fig 4.. Display settings of the output window.
- ✓ Connect digital input 5,6,7 to GND.
- ✓ Connect digital input 5 to 5V.
- ✓ Connect digital input 5 to GND.

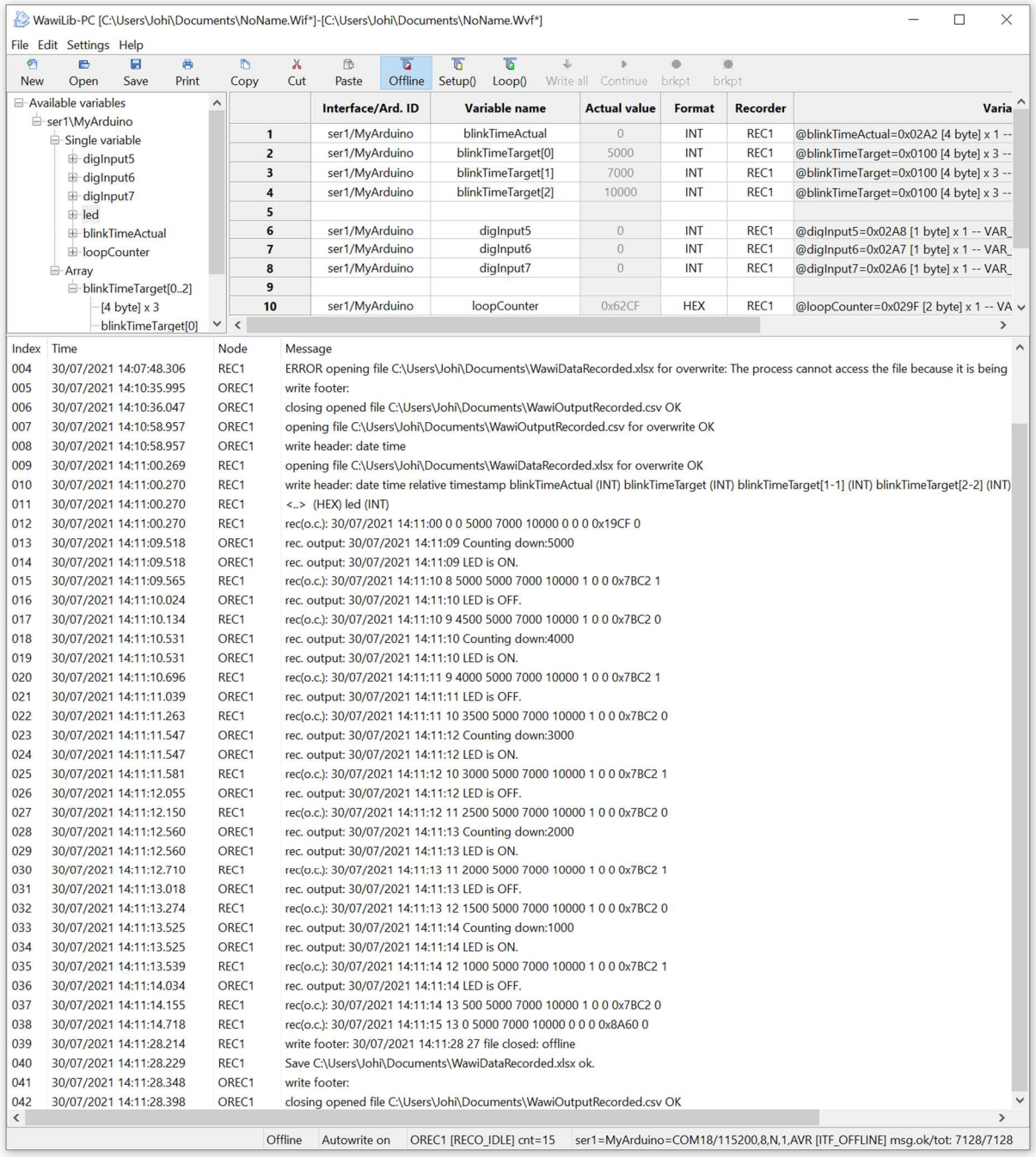


Fig 4.4. Output recording working with Display of messages in output window (facultative).

- ✓ Wait 10 seconds.
- ✓ Press “Offline”.
- ✓ Open the file with recorded output data (see bottom line in the output window fig 4.4)

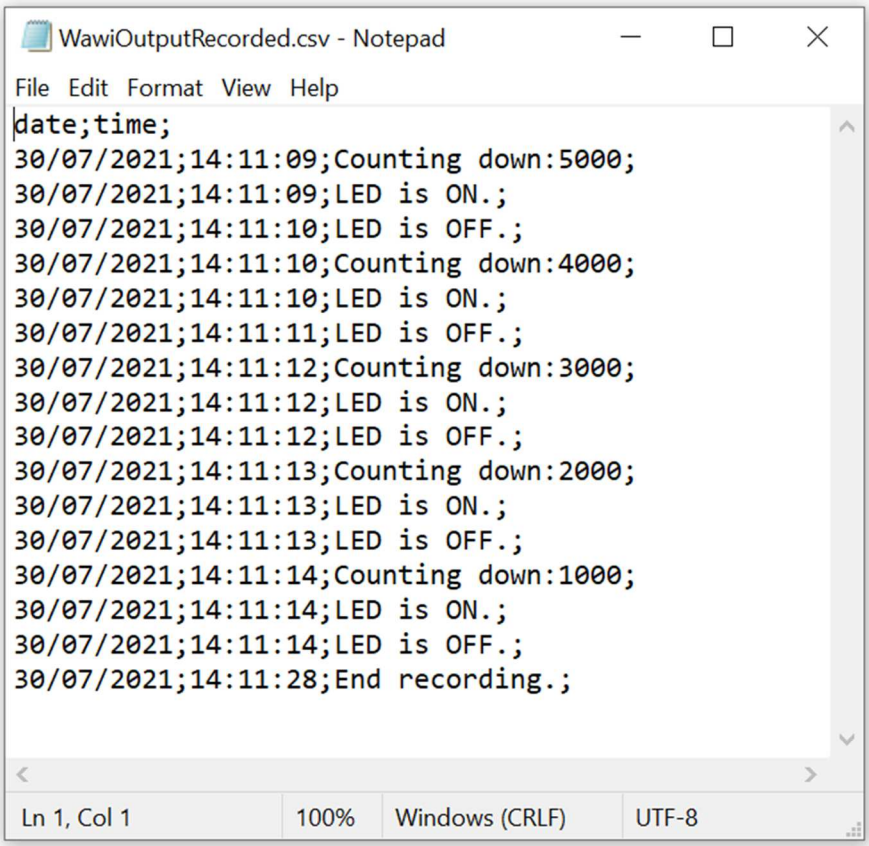


Fig 4.5. Output of .print() messages in the sketch recorded in a disk file.

Note: the data in the file was created with the statements in fig. 4.6. marked in yellow.

```
/*
* Project Name: WawiRecUsb
* File: WawiRecUsb.ino
*
* Detailed manual:
* www.SylvesterSolutions.com\documentation -> "Recording variables with WawiLib.pdf"
*
* Description: demo file library for WawiSerialUsb library.
* Data recorder demo.
* => Record values of variables to disk
* => Record .print() output to disk
* Use the USB programming port to make connection with the Arduino board.
* Variables can be checked & modified with the WawiLib-PC software.
*
* Author: John Gijs.
* Created March 2020
* More info: www.sylvestersolutions.com
* Technical support: support@sylvestersolutions.com
* Additional info: info@sylvestersolutions.com
*/

#include <WawiSerialUsb.h>

WawiSerialUsb WawiSrv;
#define LED 13 // blinking light
#define IN_5 5 // light start blinking switch 1
```

```
#define IN_6 6 // light start blinking switch 2
#define IN_7 7 // light start blinking switch 3

// variables for demo:
long int blinkTimeActual = 0; // counter blink active (milliseconds)
long int blinkTimeTarget[] = { 5000, 7000, 10000 }; // bug 1: should be { ..., ...,
10000};

bool digInput5; // state of digital input 5
bool digInput6; // state of digital input 6
bool digInput7; // state of digital input 7
bool led; // state of led
int loopCounter;

// make variables of interest know to WawiLib:
void wawiVarDef()
{
    WawiSrv.wawiVar(digInput5);
    WawiSrv.wawiVar(digInput6);
    WawiSrv.wawiVar(digInput7);
    WawiSrv.wawiVar(led);
    WawiSrv.wawiVar(blinkTimeActual);
    WawiSrv.wawiVar(loopCounter);
    WawiSrv.wawiVarArray(blinkTimeTarget);
}

void setup()
{
    Serial.begin(115200);
    // initialize WawiLib library:
    WawiSrv.begin(wawiVarDef, Serial, "MyArduino");
    pinMode(LED, OUTPUT);
    pinMode(IN_5, INPUT);
    pinMode(IN_6, INPUT);
    pinMode(IN_7, INPUT);

    WawiSrv.wawiBreakDisable();
}

void loop()
{
    digInput5 = digitalRead(IN_5);
    digInput6 = digitalRead(IN_6);
    digInput7 = digitalRead(IN_7); // bug 2: should be digInput7 = ...


    if (digInput5)
        blinkTimeActual = blinkTimeTarget[0]; // bug 3: should be activeMsSetpoint[0]

    if (digInput6)
        blinkTimeActual = blinkTimeTarget[1];

    if (digInput7)
        blinkTimeActual = blinkTimeTarget[2];

    if (digInput5 || digInput6 || digInput7)
    {
        WawiSrv.wawiBreak(1, "breakpoint after write to activeMsCounter hit");
    }

    while (blinkTimeActual > 0) // bug 4: should be activeMsCounter > 0
    {
```

```
        WawiSrv.wawiBreak(2, "In while loop");

        WawiSrv.print("Counting down:");
        WawiSrv.println(blinkTimeActual);

        WawiSrv.println("LED is ON.");
        led = HIGH;
        digitalWrite(LED, led);
        WawiSrv.delay(500);
        blinkTimeActual = blinkTimeActual - 500;

        WawiSrv.println("LED is OFF.");
        led = LOW;
        digitalWrite(LED, led);
        WawiSrv.delay(500);
        blinkTimeActual = blinkTimeActual - 500;
    }
    WawiSrv.loop();
    loopCounter++;
}
```

Fig 4.5. WawiRecUpb creating output to be recorded on a disk file.

5. Final notes

An important aspect is the use of PC memory for temporary storage of recorded data by the data recorders.

If you choose .xlsx as a data format, all the recorded data is stored into PC memory until the recording is ended. At that time a series of files is written to disk. An .xlsx file is in fact a zipped combination of xml coded files. The files contain data and references in xml format. So, storage in memory is an aspect that is linked to the concept of .xlsx files, there is no other way.

.csv files and .xml file recording works differently: data is written to the disk memory cache each time an additional record of data is recorded. This means that at a failure of the power to the PC, you will lose some records but a part of the .csv file could remain intact. For .xml files, data is written to the disk cache as well but in case of abnormal termination, the final closing records of the .xml file will not be added, so the file becomes corrupt. (You can try to recover a corrupted xml file using a text editor)

In order to minimize the risk of data loss, you can decide to create a new file every 15 minutes to make sure this data is saved to disk in case of power failure.

Another relevant aspect of data storage is the fact that recorders will start only if all of their tags can be read. When a recorder starts, it waits until all its tags are read once before it starts to record. This means that in case of an illegal/missing tag configuration, the recorder will continue to wait. You can observe the status of the data recorders in the bottom line of WawiLib-PC: REC_WAIT_FOR_TAG_READ_ONCE is the state of the recorder used to wait for successful read of all tags.

The best way to check if a recorder is started properly is to activate data recorder tracing and to look at the output window. If recorded data is appearing in the output window, it is working properly.

6. Further reading

This demo demonstrates how to record variables with WawiLib. Recording can be done time-based or on-change. Data files can have .csv or .xlsx format. File sizes can be limited by automatically restarting with a new file every 15 minutes, every hour or every day. Disk usage can be limited for each datalogger. WawiLib can clean up its data files automatically to prevent disk space usage overload.

This demo also shows how to record the output of your .print() statements used in a sketch. Also there the output of the statements can be sent to files of different types and sizes.

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