SENSIRION
THE SENSOR COMPANY

Application Note for Liquid Flow Sensors

# Performing Basic Measurements using the RS485 Sensor Cable


## Summary

This document describes the program sequences required to perform basic measurements with the RS485 Sensor Cable and how to interpret the obtained results. The modes of operation discussed in this document include Single Flow Rate Measurement, Continuous Flow Rate Measurement, and Total Volume Measurement.


## Introduction

The general working principle of the Sensirion Liquid Flow Sensors is described in the Operating Guidelines to the FlowMeterKit for Liquid Flow Products (LQ_DS_FlowMeterKit_OperatingGuidelines_EN_xx_D1).

### Communication Hardware

Typical hardware configurations for use with the RS485 Sensor Cable include:

- PC with RS485 PCI board
- PC with USB to RS485 converter
- PC with RS232 to RS485 converter
- Microcontroller with UART (Universal Asynchronous Receiver/Transmitter) interface and RS485 transceiver
- PC with USB slot (when using the cable with the integrated USB-to-RS485 converter)

The RS485 Sensor Cable is available with 3 different connector options:

- RS485 side with open wire ends, article code 1-100804-01
- RS485 side with D-sub DE-9 connector and external power supply, article code 1-100839-01
- Cable with integrated USB-to-RS485 converter, article code TBD

### RS485 Interface

The RS485 Interface uses the Sensirion High-Level Data Link Control (SHDLC) protocol for communication.

For details on how to implement the low-level SHDLC protocol (e.g. on a microcontroller) see the following documents:

- Implementation Guide to the SHDLC Protocol for the RS485 Sensor Cable (LQ_AN_RS485SensorCable_ImplementationGuideToSHDLC_EN_xx_D2)
- SHDLC Command Reference for the RS485 Sensor Cable (RS485_Sensor_Cable_SHDLC_Commands_EN_xx_D1).

For high-level implementation on Microsoft Windows Systems, Sensirion provides C driver DLLs for the RS485 Sensor Cable which may be referenced from standard programming languages. See the RS485 Sensor Cable DLL Documentation (LQ_CO_RS485_SensorCable_DLL_Documentation) for details. The DLL is available for 32 bit and 64 bit operating systems. (LQ_CO_RS485_SensorCable_DLL_Driver_EN_xx_D2, LQ_CO_RS485_SensorCable_DLL_Driver-64bit_EN_xx_D2)

Separate programming examples are available for C++, C#, and Python. Additionally, examples for LabVIEW are available, including a library which implements the most important commands.

This document describes the program sequences in a generalized form, regardless of the specific implementation.

# Single Measurement Mode

A single flow rate measurement consists of two steps: 1) starting the measurement on the sensor and 2) retrieving the measured flow rate from the sensor. The sensor output is a 16 bit (2 byte) integer, either signed (-32768...32767) or unsigned (0...65535), depending on the sensor and the measurement type. This integer linear flow data is referred to as *flow_ticks*. To obtain a flow rate value in physical units the flow_ticks need to be divided by sensor-specific *scale factor*. See also the Operating Guidelines to the FlowMeterKit for details.

## Collecting Measurement Data

*Initialization:*

- Reset the cable and sensor (recommended):
  - `DeviceReset()`
  - wait 250 milliseconds (ms) to let the reset finish
- Initialize the cable and sensor (necessary once after every power-up or device reset)

  only for SLQ-QTxxx sensors:
  - `SetHeaterMode(Heatermode = 1)` # set the heater mode to 1 (always on)

  (SLI-xxx and SLG64-xxx sensors are shipped with 'heater mode = always on' as default setting.)

  For all sensors (optional):
  - `SetResolution(resolution)` # set bit resolution of measurement (from 9 to 16bit)
- Read the scale factor, the flow unit and the measurement data type from the sensor and store them in variables (necessary once for each sensor)
  - `scalefactor = GetScaleFactor()` # read the scale factor from the sensor and store it
  - `flowunit = GetFlowUnitString()` # read the flow unit frrom the sensor and store it
  - `measurementdatatype = GetMeasurementDataType()` # read the measurement data type (signed or unsigned) from the sensor and store it.

*Perform a single measurement:*

- `StartSingleMeasurement()` # starts the measurement on the Sensor
- wait for the measurement to finish (1 to 80 ms, depending on the selected resolution)
- `sensor_output = GetSingleMeasurement()`

Use the variable `sensor_output` according to your needs, see below.

## Converting Sensor Output to Physical Flow Rate

For the interpretation of the sensor output it is necessary to distinguish the different types of implementation.

- If the low-level SHDLC protocol is implemented directly (e.g. on a microcontroller), the value returned by the function `GetSingleMeasurement()` needs to be interpreted as a signed or unsigned 16bit integer, depending on the `measurementdatatype`.
- If the windows DLL is used, the interpretation as signed or unsigned integer is handled by the DLL if the correct function `GetSingleMeasurementSigned()` or `GetSingleMeasurementUnsigned()` is used, depending on the `measurementdatatype` being signed or unsigned, respectively.

*SHDLC:*

The `sensor_output` needs to be converted to `flow_ticks` by the two's complement convention if necessary. For details, see the Implementation Guide to SHDLC.

*Windows DLL:*

Call the corresponding functions to obtain the flow_ticks:

```
if measurementdatatype == 0:          # signed
    flow_ticks = GetSingleMeasurementSigned()
else:                                 # unsigned
    flow_ticks = GetSingleMeasurementUnsigned()
```

The next step is converting the `flow_ticks` to a physical flow rate:

```
physical_flow = flow_ticks / scalefactor
```

where `physical_flow` is in the flow rate in the flow units of the sensor.

*Examples:*

Let's consider an SLQ-QT105 sensor. The scale factor of the SLQ-QT105 is '13' (subject to change), the flow unit is 'ul/s' (microliters per second) and the measurement data type is 'signed'.

*DLL implementation, measured flow rate 1000 ul/s:*

```
flow_ticks = GetSingleMeasurementSigned()
```

⇒ flow_ticks = 13000

⇒ physical_flow = 13000 / 13 = 1000

*DLL implementation, measured flow rate -500 ul/s:*

```
flow_ticks = GetSingleMeasurementSigned()
```

⇒ flow_ticks = -6500

⇒ physical_flow = -6500 / 13 = -500

# Continuous Measurement Mode

The RS485 Sensor Cable can autonomously perform single measurements at a configurable sampling time. The results of these single measurements are stored in the measurement buffer.

## Flow Chart of the Continuous Measurement Mode

The internal sequence of the continuous measurement mode is depicted in the flow chart of Fig. 1**:**

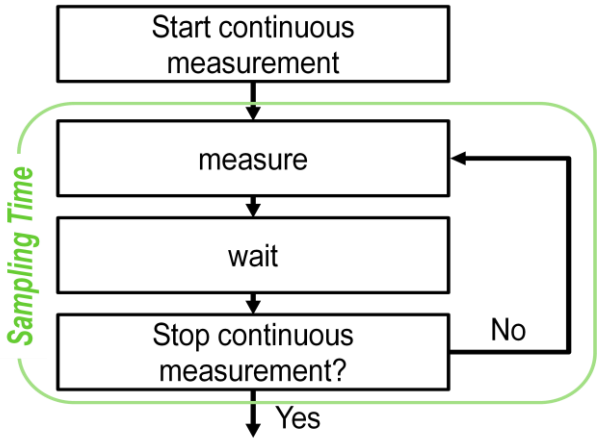



**Fig. 1:** Flow Chart of Continuous Measurement Mode

| Resolution | duration of one single measurement |
|---|---|
| 9 bit | 1 ms |
| 10 bit | 2 ms |
| 11 bit | 3 ms |
| 12 bit | 6 ms |
| 13 bit | 10 ms |
| 14 bit | 20 ms |
| 15 bit | 40 ms |
| 16 bit | 80 ms |

**Tab. 1:** Resolution vs. measurement time

The duration of the measurement depends on the selected bit resolution, see Tab. 1. Obviously, the sampling time cannot be shorter than the duration of the measurement at the selected resolution. When a single measurement is complete, the sensor cable waits in an idle state until it starts the next single measurement. The sampling time is specified in milliseconds, the maximum value being 65535. (slightly more than a minute). A sampling time of 0 (zero) may be specified. In this case the sensor measures as fast as is possible for the given data resolution, which will be slightly faster than the values stated in Tab. 1, but not as well-defined.

*Example*: Assume the resolution is 13 bit and the sampling time is 20 ms. The sensor will measure for 10ms and wait idle for another 10 ms, before starting the next single measurement.

While the measurement is running, the continuous measurement mode cannot be stopped. The sensor may therefore stay busy after receiving the `StopContinuousMeasurement()` command for up to 80 ms (in the case of 16 bit resolution), until the running single measurement is complete. If the command `StopContinuousMeasurement()` is received during the idle phase the continuous measurement mode is stopped immediately.

The data collected in Continuous Measurement Mode is stored in two ways on the RS485 Sensor Cable:

- The latest measurement is stored in a variable, 'last measurement'
- All measurements are stored in a buffer, 'measurement buffer'

Both of these may be accessed independently.

## Collecting Measurement Data

*Initialization:*

- Initialize the sensor in the same way as for a single measurement.

*Start the continuous measurement mode:*

- `StartContinuousMeasurement(samplingtime)` # starts measurement with desired sampling time

*Read the flow rate data from the sensor:*

Read only the latest measured flow value (e.g. for a live display of the flow rate) and store it in the variable `lastflow`

- `lastflow = GetLastMeasurementWithoutClear()`

or (recommended):

read the measurement buffer (to store the complete flow rate profile) and append it to the array `flowbuffer`

- `flowbuffer.append(GetMeasurementBuffer())`

*Stop Measurement:*

- `StopContinuousMeasurement()`

Use the variables `lastflow` and `flowbuffer` according to your needs.

## Converting Sensor Output to Physical Flow Rate

The variable `lastflow` and each entry in the array `flowbuffer` is the result of a single measurement. Hence, the interpretation as signed / unsigned integers in case of the SHDLC implementation (or the use of the functions with suffixes `...Signed` and `...Unsigned` in the case of the DLL implementation) is the same as discussed above for the single measurement. Also the conversion of the flow ticks to physical flow rate values is the same.

## Working with the Last Measurement

The latest measurement is stored in a separate variable on the RS485 Sensor Cable. This variable is independent of the measurement buffer (see below). It can be accessed by two different functions:

- `GetLastMeasurement()`. If the last measurement is read out with this function, its value is cleared after readout. If the function is called another time before a new measurement has been acquired, the function will return no value.

and

- `GetLastMeasurementWithoutClear()`. The last measurement can be read out at any time with this function. After read out, the variable on the Sensor Cable is not cleared. Calling the GetLastMeasurementWithoutClear() function repeatedly will return the same measurement value until a new measurement is available.

The last measurement can be used to get the flow rate currently measured by the flow meter.

## Working with the Measurement Buffer

The measurement buffer is a first-in-first-out (FIFO) buffer on the RS485 Sensor Cable in which the single measurement results collected in continuous measurement mode are stored. In normal operation, the buffer can hold up to 127 entries. (A more complicated extended buffer mode allows handling up to 1000 entries. For most applications 127 entries are sufficient. Contact Sensirion if your application requires working with the extended buffer.)

The command `GetMeasurementBuffer()` reads all available measurements from the measurement buffer and clears the measurement buffer on the RS485 Sensor Cable. If 127 values are returned, the buffer was full and probably measurements have been lost since the last call to `GetMeasurementBuffer()`. If this happens, read the buffer more frequently.

In the array returned by `GetMeasurementBuffer()` the first entry corresponds to the oldest measurement in the data packet, the last entry corresponds to the newest measurement. The data packets may be concatenated on the controller system to reproduce the entire flow rate profile.

*Example:*

```
flowbuffer = [ ]                          # initialize an array to store the measurement values

measbuffer = GetMeasurementBuffer()
⇒ measbuffer = [11, 22, 33, 44]           # example
flowbuffer.extend(measbuffer)             # append measbuffer to flow buffer
⇒ flowbuffer = [11, 22, 33, 44]

measbuffer = GetMeasurementBuffer()
⇒ measbuffer = [55, 66, 77]               # example
flowbuffer.extend(measbuffer)             # append measbuffer to flow buffer
⇒ flowbuffer = [11, 22, 33, 44, 55, 66, 77]

measbuffer = GetMeasurementBuffer()
⇒ measbuffer = [88, 99, 1010, 1111]       # example
flowbuffer.extend(measbuffer)             # append measbuffer to flow buffer
⇒ flowbuffer = [11, 22, 33, 44, 55, 66, 77, 88, 99,1010, 1111]
```

## Timing Information in Continuous Measurement Mode

In continuous measurement mode, the sensor cable performs single measurements to the measurement buffer at a regular time interval, defined as the sampling time. However, the on-board electronics of the RS485 Sensor Cable do not include a very precise internal clock, therefore clock speeds vary slightly from cable to cable. Although these variations are typically below 0.1%, when running the continuous measurement mode for a prolonged time (several minutes, hours, or days) the total number of samples acquired within a fixed amount of time will therefore vary from cable to cable.

If an absolute time stamp is required for each entry in the measurement buffer, it is recommended to reference each reading of the measurement buffer to the controller system time by the following procedure:

- `measurementtimes = [ ]` # initialize an array to store the time stamps
- `flowbuffer = [ ]` # initialize an array to store the measurement values

then on each reading of the measurement buffer do the following:

- `currentsystemtime = ...` # get the current system time of the controller
- `measbuffer = GetMeasurementBuffer()` # immediately after, read the measurement buffer
- `measbufferlength = length(measbuffer)` # get the number of entries in measbuffer

compute the timestamp of the first entry in the measurement buffer:

- `timestamp = currentsystemtime – (measbufferlength * samplingtime)`

where `samplingtime` is the sampling time of the continuous measurement mode.

then fill the `measuretimes` array based on this first timestamp:

- ```
  for i from 1 to measbufferlength:
      measuretimes.append(timestamp)
  ```
  ```

```
            timestamp = timestamp + samplingtime
```

- `flowbuffer.extend(measbuffer)` # append the measbuffer to the flowbuffer

The above procedure is illustrated by the following example:

*Example:*

samplingtime = 0.02 # 20 milliseconds = 0.02 seconds

*# first reading of the measurement buffer*
currentsystemtime = 1.163 # system time in seconds with milliseconds, example
`measbuffer = GetMeasurementBuffer()`
⇒ measbuffer = [11, 22, 33, 44, 55, 66] # example
⇒ measbufferlength = 6
⇒ first timestamp = 1.163 – 6*0.02 = 1.043
⇒ measuretimes = [1.043, 1.060, 1.083, 1.103, 1.123, 1.143]
⇒ flowbuffer = [11, 22, 33, 44, 55, 66]

*# second reading of the measurement buffer*
currentsystemtime = 1.227 # system time in seconds with milliseconds, example
`measbuffer = GetMeasurementBuffer()`
⇒ measbuffer = [77, 88, 99] # example
⇒ measbufferlength = 3
⇒ first timestamp = 1.227 – 3*0.02 = 1.167
⇒ measuretimes = [1.043, 1.060, 1.083, 1.103, 1.123, 1.143, 1.167, 1.187, 1.207]
⇒ flowbuffer = [11, 22, 33, 44, 55, 66, 77, 88, 99]

The timestamps in `measuretimes` correspond to the entries in the `flowbuffer` as follows:

| measurement | 11 | 22 | 33 | 44 | 55 | 66 | 77 | 88 | 99 |
|---|---|---|---|---|---|---|---|---|---|
| timestamp | 1.043 | 1.063 | 1.083 | 1.103 | 1.123 | 1.143 | 1.167 | 1.187 | 1.207 |
| difference of timestamps | | 0.020 | 0.020 | 0.020 | 0.020 | 0.020 | **0.024** | 0.020 | 0.020 | |

As can be seen from the above table, the procedure suggested in this section introduces a jitter of up to ±1 sampling time between data packets corresponding to different readings of the measurement buffer. On the other hand it ensures that the absolute timing information remains correct with respect to the controller system clock (to within ±1 sampling time).

As a general rule it is recommended that the sampling time should be less than half of the time scale of signal variations which are of interest. The jitter of ±1 sampling period is therefore generally acceptable. For instance, if typical flow rate variations are expected on the time scale of 50 ms, a sampling period of 20 ms is reasonable and an absolute timing error of ±20 ms is acceptable.

# Total Volume Measurement using the Totalizator

The RS485 Sensor cable provides an internal Totalizator (Totalizer) which allows adding the flow rate values measured in continuous measurement mode. In combination with a fixed sampling time, the value of the Totalizator can be used to compute the total volume measured by the sensor since the last reset of the Totalizator. The Totalizator may be enabled, disabled or reset by the user at any time. The Totalizator is only available in continuous measurement mode and the sampling time should be explicitly specified. The totalizator value is stored as an 64 bit integer value which is large enough to avoid any problems with overflow.

## Collecting Measurement Data

*Initialization:*

- Initialize the sensor in the same way as for a continuous measurement.

*At any time, when you want to start integration, enable the Totalizator and reset its value:*

- `SetTotalizatorStatus(Status = 1)` # enable the Totalizator
- `ResetTotalizator()` # reset the Totalizator value to zero

*Start the continuous measurement mode:*

- `StartContinuousMeasurement(samplingtime)` # starts measurement with desired sampling time

*At any time you may read the totalizator value:*

- `totalizator_output = GetTotalizatorValue()` # read the Totalizator value and store it

*At any time you may reset the totalizator value to zero:*

- `ResetTotalizator()` # reset the Totalizator value to zero

Use the variable `totalizator_output` according to your needs, see below.

## Computing the Physical Volume from the Totalizator Output

*SHDLC:*

The `totalizator_output` needs to be converted to `totalizator_ticks` by the two's complement convention. See the Implementation Guide to SHDLC.

*Windows DLL:*

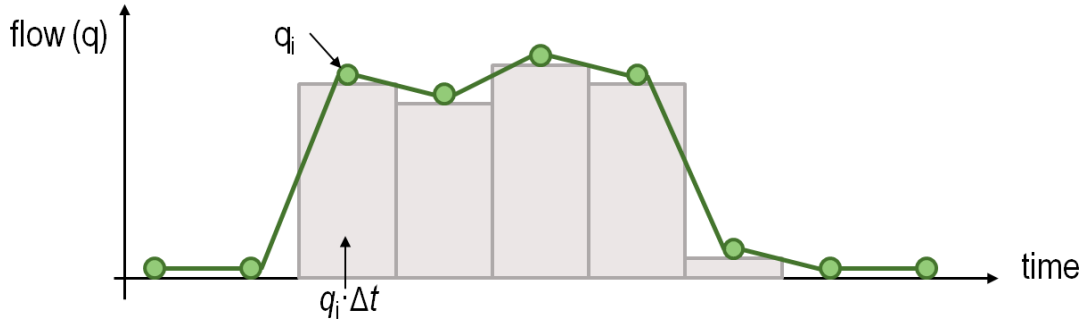
The sign conversion is handled by the DLL, so:

```
totalizator_ticks = GetTotalizatorValue()
```

The signed integer values correspond to the added sensor output values. These need to be converted to physical flow by dividing through the scale factor, as above for the flow rates:

```
totalizator_ticks_scaled = totalizator_ticks / scalefactor
```

This sum of physical flow values must be converted to a volume by taking into account the time spacing between two subsequent flow values, see the following viewgraph.



During a dispense, the sensor measures flow rate values $q_i$ (green dots) at a specified time interval $\Delta t$ (the sampling time). In order to obtain the volume of the dispense, the area under the curve must be determined. This is equivalent to adding the grey rectangles in the viewgraph. Each rectangle represents a small volume $V_i$, and has width and height $\Delta t$ and $q_i$ respectively.

The total physical volume is the sum of these physical volumes $V_i$, where for each volume $V_i = q_i \cdot \Delta t_i$. Therefore the total volume can be computed as follows: $V = \Sigma V_i = \Sigma q_i \Delta t_i = \Sigma q_i \Delta t = \Delta t \, \Sigma q_i$.

The value `totalizator_ticks_scaled` as computed above from the `totalizator_output` is a sum of physical flow rate values: $\Sigma q_i$

The time interval $\Delta t$ is the time between measurements (the sampling time), in the same time units as the physical flow unit of the sensor.

*Example 1:* sampling time 20 ms, flow unit is ul/sec, i.e. $\Delta t$ = 0.02 sec

*Example 2:* sampling time 20 ms, flow unit is ml/min, i.e. $\Delta t$ = 0.000333 min

The physical volume can therefore be calculated as follows:

```
physical_volume = totalizator_ticks / scalefactor * sampling_time
```

*Example:*

SLQ-QT105 Sensor, measured volume: 300 ul. The scale factor of the SLQ-QT105 is 13 (subject to change), flow unit is ul/s (microliters per second), measurement interval is 20 ms. When using the DLL, we have

```
totalizator_ticks = GetTotalizatorValue()
```
⇒ totalizator_ticks = 195000 # example
⇒ totalizator_ticks_scaled = 195000 / 13 = 15000
⇒ physical_volume = 15000 * 0.02 = 300

the measured volume is therefore 300 microliters.

## Headquarter and Sales Offices


SENSIRION AG
Laubisruetistr. 50
CH-8712 Staefa ZH
Switzerland
Phone: + 41 (0)44 306 40 00
Fax: + 41 (0)44 306 40 30
info@sensirion.com
www.sensirion.com

SENSIRION Inc
Westlake Pl. Ctr. I, suite 204
2801 Townsgate Road
Westlake Village, CA 91361
USA
Phone: +1 805-409 4900
Fax: +1 805-435 0467
michael.karst@sensirion.com
www.sensirion.com

SENSIRION Japan
Sensirion Japan Co. Ltd.
Shinagawa Station Bldg. 7F
4-23-5 Takanawa
Minato-ku, Tokyo, Japan
Phone: +81 3-3444-4940
Fax: +81 3-3444-4939
info@sensirion.co.jp
www.sensirion.co.jp

SENSIRION Korea Co. Ltd.
#1414, Anyang Construction Tower B/D,
1112-1, Bisan-dong, Anyang-city,
Gyeonggi-Province, South Korea
Phone: +82-31-440-9925~27
Fax: +82-31-440-9927
info@sensirion.co.kr
www.sensirion.co.kr

SENSIRION China Co. Ltd.
Room 2411, Main Tower
Jin Zhong Huan Business Building,
Postal Code 518048
Futian District, Shenzhen, PR China
Phone: +86 755 8252 1501
Fax: +86 755 8252 1580
info@sensirion.com.cn/
www.sensirion.com.cn


Find your local representative at: www.sensirion.com