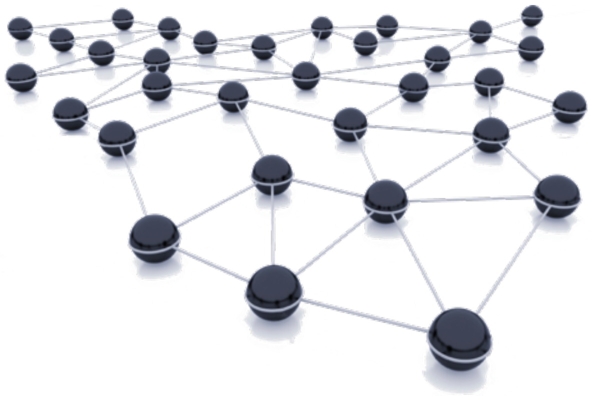
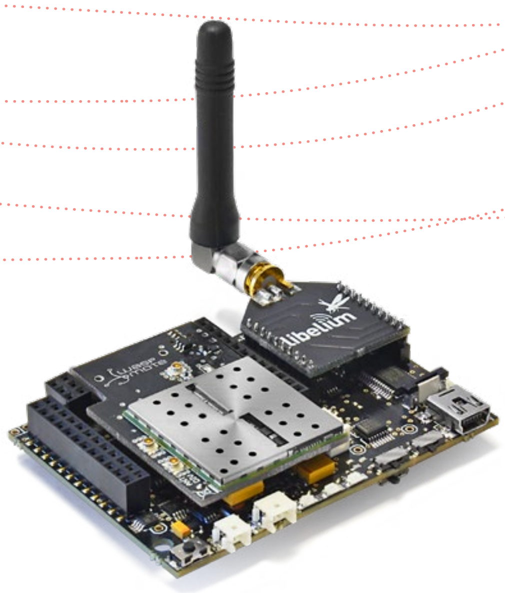
Waspmote Data Frame

Programming Guide





Document Version: v7.6 - 02/2019

INDEX

1. Introduction ... 3

- 1.1. Waspmote Frame files ... 3
- 1.2. Constructor ... 3
- 1.3. Library functions ... 3
- 1.4. Predefined constants ... 4

2. Frame structure ... 5

- 2.1. ASCII frame ... 5
  - 2.1.1. ASCII header ... 6
  - 2.1.2. ASCII payload ... 7
- 2.2. Binary frame ... 8
  - 2.2.1. Binary header ... 8
  - 2.2.2. Binary payload ... 9
- 2.3. Tiny frame ... 11
- 2.4. Frame types ... 12
- 2.5. Sensor fields ... 13

3. Frame Functions ... 30

- 3.1. Setting the Waspmote Identifier ... 30
- 3.2. Creating new frames ... 30
- 3.3. Setting the frame size ... 31
- 3.4. Setting the frame type ... 33
- 3.5. Adding sensor fields ... 34
- 3.6. Adding new sensor types ... 35
- 3.7. Showing the current Frame ... 35
- 3.8. Frame fragmentation (only binary frames) ... 35

4. Encrypted frames ... 37

- 4.1. Encrypted frame format ... 37
- 4.2. Device to gateway encryption ... 37
- 4.3. Device to Cloud encryption ... 39

5. Code examples ... 40

6. API changelog ... 41

7. Documentation changelog ... 42

1. Introduction

This guide explains the Waspmote Frame features and functions. There are some variations in this library for our new product lines Waspmote v15 and Plug & Sense! v15, released on October 2016.

Some functions may not be compatible with Waspmote v12 or Plug & Sense! v12. Besides, old functions may no longer exist. If you are using previous versions of our products, please use the corresponding guides, available on our Development website.

You can get more information about the generation change on the document "New generation of Libelium product lines".

Differences of this library compared to the previous version:

- New sensor field table and definitions which most of them are not compatible with former version
- Some old sensor field definitions have been deprecated for Waspmote v15 because they are no longer needed
- New frame type is used for both Binary and ASCII frame in Waspmote v15. Only new Meshlium devices will be able to receive frames from the new lines Waspmote v15 and Plug & Sense! v15
- Some frame types have been deprecated because they are no longer needed

1.1. Waspmote Frame files

WaspFrame.h, WaspFrame.cpp, WaspFrameConstantsv12.h, WaspFrameConstantsv15.h

It is mandatory to include the WaspFrame library when using this class. The following line must be introduced at the beginning of the code:

```
#include <WaspFrame.h>
```

Libelium recommends the use of the official Data Frame format, explained in this guide. It is especially good for the projects with a Meshlium, because it can parse frames in an automatic way thanks to the feature "Sensor Parser".

1.2. Constructor

To start using the Waspmote Frame library, an object from the `WaspFrame` class must be created. This object, called `frame`, is created inside the Waspmote Frame library and it is public to all libraries. It is used through the guide to show how the Waspmote Frame library works.

When creating this constructor, some variables are defined with a value by default.

1.3. Library functions

Through this guide there are many examples of the WaspFrame class usage. In these examples, library functions are called to execute the commands, storing in their related variables the parameter value in each case.

Example of use:

```
{
  frame.createFrame(); // create a new frame
}
```

1.4. Predefined constants

There are some predefined constants in a file called ‘WaspFrame.h’. These constants define some parameters like the maximum size of each frame:

MAX_FRAME: (default value 255) specifies the maximum size of the frames to be created.

ASCII: this constant is used to define an ASCII frame mode.

BINARY: this constant is used to define a Binary frame mode.

ENCRYPTED_FRAME: this constant is used to define an encrypted frame.

Besides, there are sensor TAGs defined for each kind of sensor. These labels are used to set different fields inside the frame in order to distinguish between different sensor values and identify them.

2. Frame structure

The Waspmote Frame was designed in order to create sensor data frames with a specific format. This data protocol is supported by Meshlium (Meshlium can decode these data frames), so this is the format to be used in order to transmit data to Meshlium.

There are two kinds of frames: ASCII and Binary.

Besides, a special frame format was designed in order to send sensor data via low bit-rate protocols with short payload size. This frame type is called 'Tiny' frame. The user must keep in mind that this protocol is not integrated into Meshlium (in fact, this frame type is mainly designed for constrained radios like Sigfox or LoRaWAN, and when operating with these protocols the receiver is not Meshlium, but a Sigfox or LoRaWAN base station).

2.1. ASCII frame

These frames are supposed to facilitate the comprehension of the data to be sent. As the frame is composed of ASCII characters is easier to understand all the fields included within the payload.

It is possible to identify two different parts inside the frame. The first one corresponds to the header and its structure is always the same. The second one corresponds to the payload and it is where the sensor values are included.

The following figure describes the ASCII Frame structure:

HEADER										PAYLOAD						
<=>	Frame Type	Num Fields	#	Serial ID	#	Waspmote ID	#	Sequence	#	Sensor_1	#	Sensor_2	#	...	Sensor_n	#

Figure: ASCII Frame structure

2.1.1. ASCII header

The structure fields are described below with an example:

HEADER										PAYLOAD					
<=>	0x80	0x03	#	35690284	#	NODE_001	#	214	#	BAT:35	#	GPS:31.200;42.100	#	DATE:12-01-01	#
A	B	C	D	E	D	F	D	G	D	sensor1	D	sensor2	D	sensor3	D

Figure: ASCII Frame example

A → Start Delimiter [3 bytes]: It is composed of three characters: "<=>". This is a 3-byte field and it is necessary to identify each frame starting.

B → Frame type byte [1 byte]: This field is used to determine the frame type. There are two kinds of frames: Binary and ASCII. But it also defines the aim of the frame such event frames or alarm frames. This field will be explained in the following sections.

C → Number of Fields [1 byte]: This field specifies the number of sensor fields sent in the frame. This helps to calculate the frame length.

D → Separator [1 byte]: The '#' character defines a separator and it is put before and after each field of the frame.

E → Serial ID [16 bytes]: This is a 16-byte field which identifies each Waspmote device uniquely. The serial ID is taken from a specific chip integrated in Waspmote that gives a different identifier to each Waspmote device. So, it is only readable and it can not be modified.

F → Waspmote ID [0..16 bytes]: This is a string defined by the user which may identify each Waspmote inside the user's network. The field size is variable [from 0 to 16 bytes]. When the user do not want to give any identifier, the field remains empty between frame's separators: "##".

G → Frame sequence [1..3 bytes]: This field indicates the number of sequence frame. This counter is 8-bit, so it goes from 0 to 255. However, as it is an ASCII frame, the number is converted to a string in order to be understood. This is the reason the length of this field varies between one and three bytes. Each time the counter reaches the maximum 255, it is reset to 0. This sequence number is used in order to detect loss of frames.

Note: *There is only one frame counter, so in the case two communication modules are used, this counter is incremented each time a new frame is created. If each module needs to create a new frame, the counter will be incremented by 2 in the same loop, one for each frame creation.*

2.1.2. ASCII payload

The frame payload is composed of several sensor data. All data sent in these fields correspond to a predefined sensor data type in the sensor table. This sensor table is stored in Meshlium (gateway of the network) and it will be used in order to interact with the database.

ASCII frame payload					
Sensor_1	#	Sensor_2	#	Sensor_n	#

Figure: ASCII payload structure

There are three types of ASCII sensor data:

- **Simple Data:** The sensor field is composed of a single sensor value. The format is: "<sensor_id>:<value>" and a separator character [#] is set at the end of the value. For example, a temperature field indicating 23ºC would be as follows:

```
TC:23#
```

- **Complex Data:** This is the format used to send sensor fields composed of two or three values. The format is: "<sensor_id>:<value>;<value>;<value>" and a separator character [#] is set at the end of the last value. Accelerometer and GPS measurements are some examples:

```
ACC:996;-250;-100#
GPS:41.680616;-0.886233#
```

- **Special Data:** Date and time are defined in a special format.

  Date is defined as "`yy-mm-dd`" where:
  - yy: year
  - mm: month
  - dd: day of month

  Example: #DATE:13-01-01#

  Time is formatted as "`hh-mm-ss+GMT`" where:

- hh: hours
- mm: minutes
- ss: seconds
- GMT: GMT is added after hh-mm-ss. It is possible to avoid this information in order to save frame size.

Example without GMT:	`TIME:12-24-16#`
Example with GMT:	`TIME:12-24-16+1#`

2.2. Binary frame

This frame type has been designed to create more compressed frames. The main goal of defining binary fields is to save bytes in frame's payload in order to send as much information as possible. The main disadvantage is the legibility of the frame.

As the ASCII frames, the Binary frames are also composed of two different parts: header and payload. The header of the Binary frame is quite similar to the ASCII frame except for the frame sequence number and the separator at the end of the header.

The following figure describes the Binary Frame structure:

HEADER							PAYLOAD			
<=>	Frame Type	Num of bytes	Serial ID	Waspmote ID	#	Sequence	Sensor_1	Sensor_2	...	Sensor_n

Figure: Binary Frame structure

2.2.1. Binary header

The structure fields are described below with an example:

HEADER							PAYLOAD								
<=>	0x00	0x17	0x74F94515	NODE_001	#	0x00	ID	Byte 1	Byte 2	ID	Byte 1	Byte 2	ID	Byte 1	Byte 2
A	B	C	E	F	D	G	Sensor 1			Sensor 2			Sensor 3		

Figure: Binary Frame example

A → Start Delimiter [3 bytes]: It is composed of three characters: "<=>". This is a 3-byte field and it is necessary to identify each frame starting.

B → Frame type [1 byte]: This field is used to determine the frame type. There are two kind of frames: Binary and ASCII. But it also defines the aim of the frame such event frames or alarm frames. This field will be explained in the following sections.

C → Number of bytes [1 byte]: This field specifies the number of bytes after this field until the end of the payload is found.

D → Separator [1 byte]: The '#' character defines a separator and it is put between some fields which length is not specified. This helps to parse the different fields in reception.

E → Serial ID [8 bytes]: This is a 8-byte field which identifies each Waspmote device uniquely. The serial ID is taken from a specific chip integrated in Waspmote that gives a different identifier to each Waspmote device. So, it is only readable and it can not be modified. Note that the Serial ID is sent as a binary field too.

F → Waspmote ID [variable]: This is a string defined by the user which may identify each Waspmote inside the user's network. The field size is variable [from 0 to 16 bytes]. When the user do not want to give any identifier, the field remains empty indicated by a unique '#' character.

G → Frame sequence [1 byte]: This field indicates the number of sent frame. This counter is 8-bit, so it goes from 0 to 255. Each time it reaches the maximum 255 is reset to 0. This sequence number is used in order to detect loss of frames.

***Note:** There is only one frame counter, so in the case two communication modules are used, this counter is incremented each time a new frame is created. If each module needs to create a new frame, the counter will be incremented by 2 in the same loop, one for each frame creation.*

2.2.2. Binary payload

The frame payload might be composed of several sensor data. All data sent in these fields correspond to a predefined sensor data type in the sensor table. Regarding the binary format, each sensor in the sensor table determines the number of necessary bytes to express the sensor value. The sensor table is stored in Meshlium (gateway of the network) and it will be used in order to interact with the database.

Binary frame payload			
Sensor data 1	Sensor Data 2	...	Sensor Data n

Figure: Binary payload structure

There are three types of Binary sensor fields:

- **Simple Data:** The sensor field is composed of a single sensor value. The format of this field is: the first byte codifies the sensor identifier. Following the first byte and according to the sensor table, there is a number of bytes which correspond to the sensor value. For example, the temperature sensor is a float number, so it is a 4-byte field. Thus, the sensor field for 27ºC will be set as follows:

Sensor data (SENSOR_GAS_TC)				
Sensor ID	**Sensor field 1**			
Byte 0	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4**
0x4A	0x00	0x00	0xD8	0x41

Figure: Binary simple sensor field

Note: *Floats are codified so they are not a simple conversion.*

- **Complex Data:** This is the format used to send sensor data composed of more than one value. The format of this field is: the first byte codifies the sensor type. Then, the different values are codified using as many bytes as they specify in the sensor table. For example, the GPS field is composed of both latitude and longitude floats, which means that 8 bytes are needed for both float values:

Sensor data (SENSOR_GPS)								
Sensor ID	**Sensor field 1**				**Sensor field 2**			
Byte 0	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6**	**Byte 7**	**Byte 8**
0x35	0x59	0x9D	0x26	0x42	0xE0	0x10	0x61	0xBF

Figure: Binary complex sensor field (GPS)

Sensor data (SENSOR_ACC)						
Sensor ID	**Sensor field 1**		**Sensor field 2**		**Sensor field 3**	
Byte 0	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6**
0x3F	0x59	0x9D	0xE0	0x10	0x00	0x0A

Figure: Binary complex sensor field (accelerometer)

Note: *Floats are codified so they are not a simple conversion.*

- **String:** This is the only field that is formed differently: the first byte codifies the sensor type, the second byte defines the string length, and the rest of the bytes belong to the string itself according to the length previously defined. For example, the string “hello” is formatted as follows:

Sensor data (SENSOR_STR)						
Sensor ID	**Sensor field 1**					
Byte 0	**Byte 1 (length)**	**Byte 2 (‘h’)**	**Byte 3 (‘e’)**	**Byte 4 (‘l’)**	**Byte 5 (‘l’)**	**Byte 6 (‘o’)**
0x41	0x05	0x68	0x65	0x6C	0x6C	0x6F

Figure: Binary string sensor field

2.3. Tiny frame

This type of frame has been designed to create short frames with data. The purpose of implementing tiny frames is to be able to create sensor data frames which can be send via short-payload protocols, like Sigfox or LoRaWAN. The main disadvantage is that Meshlium does not support this frame format. However, as short-payload protocols send data directly to a Cloud system through a base station, the tiny frame format is perfect for these applications.

Tiny frames generation is based on a previously created binary frame. The goal is to create a full binary frame, and then generate different tiny frames from the original binary frame. So the steps involved in tiny frame creation are:

- Step 1: The user must create a single binary frame
- Step 2: Add sensor data as usual
- Step 3: Finally, generate tiny frames from the current contents of the binary frame

In order to generate new tiny frames the following functions are described.

The `setTinyLength()` function allows the user to configure the maximum payload of the tiny frames to be generated. By default, the maximum payload size is 12 bytes. The range of this parameter goes from 10 to 100 bytes.

The `generateTinyFrame()` function allows the user to generate a new tiny frame from an original binary frame. The contents of the tiny frame are stored in the `bufferTiny` buffer. The length of this buffer is provided by the `lengthTiny` variable. This function returns the number of pending sensor fields to be filled in a tiny frame from the original binary frame. So, this function should be called several times until no more pending fields are returned.

The following figure describes the tiny frame structure:

HEADER		**PAYLOAD**			
Sequence	Lenght	Sensor_1	Sensor_2	...	Sensor_n

Figure: Tiny Frame structure

The structure fields are described below with an example:

HEADER		**PAYLOAD**								
0x00	0x0B	ID	Byte 1	Byte 2	ID	Byte 1	Byte 2	ID	Byte 1	Byte 2
A	B	Sensor 1			Sensor 2			Sensor 3		

Figure: Tiny frame example

A → Frame sequence [1 byte]: This field indicates the sequence number. This is an 8-bit counter, so it goes from 0 to 255. Each time it reaches the maximum 255, it is reset to 0. This sequence number is used in order to detect loss of frames. All tiny frames generated from the same original binary frame have the same sequence number. So, this can be understood as frame fragmentation.

B → Length [1 byte]: This is the total number of bytes of the tiny frame including both header and payload. In other words, the total length stored in the `lengthTiny` variable.

2.4. Frame types

As it was said before, there is a specific field in the header which specifies the frame type. This field is defined by a byte noted as the sequence of the following bits: $b_7b_6b_5b_4b_3b_2b_1b_0$:

b_7: The most significant bit specifies if the frame is ASCII (b_7=1) or Binary (b_7=0).

b_6-b_0: The rest of the bits determine the frame type which might be an event frame, a time out frame, etc.

Frame Type Byte		Decimal value	Identifier	Description
bit7	bit6-bit0			
0 (Binary)	0000000	0	Information	Information frame for Waspmote v12
	0000001	1	TimeOut	Frame sent when time is out
	0000010	2	Event	Frame sent when an event occurs
	0000011	3	Alarm	Frame sent when an alarm occurs
	0000100	4	Service1	Frame for “keep alive” advertisement
	0000101	5	Service2	Frame for “low battery” advertisement
	0000110	6	Information	Information frame for Waspmote v15
	0000111	7	Information	Information frame for Smart agriculture Xtreme
	0001000	8	Information	Information frame for Smart Water Xtreme
	...	9 to 95	...	Reserved types
	1100000	96	AES_ECB_FRAME_v15	AES encrypted frame (Waspmote v15)
	1100001	97	AES128_ECB_FRAME_v12	AES-128 Encrypted frame (Waspmote v12)
	1100010	98	AES192_ECB_FRAME_v12	AES-192 Encrypted frame (Waspmote v12)
	1100011	99	AES256_ECB_FRAME_v12	AES-256 Encrypted frame (Waspmote v12)
	1100100	100	AES_ECB_END_TO_END_V15	AES encrypted frame from device to Cloud (Waspmote v15)
		101	AES_ECB_END_TO_END_V12	AES encrypted frame from device to Cloud (Waspmote v12)
	...	102 to 127	...	Reserved types
1 (ASCII)	0000000	128	Information	Information frame for Waspmote v12
	0000001	129	TimeOut	Frame sent when time is out
	0000010	130	Event	Frame sent when an event occurs
	0000011	131	Alarm	Frame sent when an alarm occurs
	0000100	132	Service1	Frame for “keep alive” advertisement
	0000101	133	Service2	Frame for “low battery” advertisement
	0000110	134	Information	Information frame for Waspmote v15
	0000111	135	Information	Information frame for Smart agriculture Xtreme
	0001000	136	Information	Information frame for Smart Water Xtreme
	...	137 to 154	...	Reserved types
	10011011	155	Time Sync	Frame for HTTP query with Time Stamp info from Meshlium
	...	156 to 255	...	Reserved types

Figure: Frame types

2.5. Sensor fields

The following table describes all possible sensor fields.

Reference: This column refers to the sensor reference given by Libelium to each sensor in the sensor catalog.

Sensor TAG: This column defines the constants needed to add each sensor to the frame using `addSensor()` function.

SENSOR ID: Each sensor field has its own identifier. Depending on the Sensor TAG chosen, a different identifier will be set as sensor identifier. ASCII frames use a string label as sensor identifier. Binary frames use a byte as sensor identifier so as to save frame size.

Number of Fields: Defines the number of different fields a sensor value presents. Most sensors only need a unique field. But there are some cases which need more than one, i.e. the GPS module which needs 2 fields for both latitude and longitude measurements.

Type and Size: Indicates the variable type which has to be used for each sensor. The possibilities are: uint8_t (1 byte), int (2 bytes), float (4 bytes), unsigned long (4 bytes), string (variable size). ASCII frames don't have constraints when adding sensor fields in order to facilitate the user to insert new sensor data.

Default Decimal Precision: Defines for each sensor the number of decimals used in ASCII frames when using float variable types.

Units: This column defines the units used for each sensor

	Sensor	Sensor Reference	Sensor TAG	SENSOR ID		Number Of Fields	Binary		ASCII	Unit
				Binary	ASCII		Type of variable	Size per Field (Bytes)	Default Decimal Precision	
Gases (Smart Environment)	Carbon Monoxide – CO	9229	SENSOR_GASES_CO	0	CO	1	float	4	3	ppm
	Carbon Dioxide – CO2	9230	SENSOR_GASES_CO2	1	CO2	1	float	4	3	ppm
	Oxygen – O2	9231	SENSOR_GASES_O2	2	O2	1	float	4	3	ppm
	Methane – CH4	9232	SENSOR_GASES_CH4	3	CH4	1	float	4	3	ppm
	Ozone – O3	9258	SENSOR_GASES_O3	4	O3	1	float	4	3	ppm
	Ammonia – NH3	9233	SENSOR_GASES_NH3	5	NH3	1	float	4	3	ppm
	Nitrogen Dioxide – NO2	9238	SENSOR_GASES_NO2	6	NO2	1	float	4	3	ppm
	Liquefied Petroleum Gases	9234	SENSOR_GASES_LPG	7	LPG	1	float	4	3	ppm
	Air Pollutants 1	9235	SENSOR_GASES_AP1	8	AP1	1	float	4	3	ppm
	Air Pollutants 2	9236	SENSOR_GASES_AP2	9	AP2	1	float	4	3	ppm
	Solvent Vapors	9237	SENSOR_GASES_SV	10	SV	1	float	4	3	ppm
	Hydrocarbons – VOC	9201	SENSOR_GASES_VOC	11	VOC	1	float	4	3	ppm
	BME – Temperature Celsius	9370-P	SENSOR_GASES_TC	74	TC	1	float	4	2	° C
	BME – Temperature Farhenheit	9370-P	SENSOR_GASES_TF	75	TF	1	float	4	2	° F
	BME – Humidity	9370-P	SENSOR_GASES_HUM	76	HUM	1	float	4	1	%RH
	BME – Pressure	9370-P	SENSOR_GASES_PRES	77	PRES	1	float	4	2	Pascales
	Luxes	9325	SENSOR_GASES_LUXES	78	LUX	1	uint32_t	4	0	luxes
	Ultrasound	9246-P	SENSOR_GASES_US	79	US	1	uint16_t	2	0	cm
Gases PRO (Smart Environment PRO)	Carbon Monoxide – CO	9371-P	SENSOR_GASES_PRO_CO	0	CO	1	float	4	3	ppm
	Carbon Dioxide – CO2	9372-P	SENSOR_GASES_PRO_CO2	1	CO2	1	float	4	3	ppm
	Oxygen – O2	9373-P	SENSOR_GASES_PRO_O2	2	O2	1	float	4	3	ppm
	Methane – CH4	9379-P	SENSOR_GASES_PRO_CH4	3	CH4	1	float	4	3	% LEL
	Ozone – O3	9374-P	SENSOR_GASES_PRO_O3	4	O3	1	float	4	3	ppm
	Ammonia – NH3	9378-P	SENSOR_GASES_PRO_NH3	5	NH3	1	float	4	3	ppm
	Nitrogen Dioxide – NO2	9376-P	SENSOR_GASES_PRO_NO2	6	NO2	1	float	4	3	ppm
	Nitrogen Monoxide – NO	9375-P	SENSOR_GASES_PRO_NO	12	NO	1	float	4	3	ppm
	Chlorine – CL2	9386-P	SENSOR_GASES_PRO_CL2	13	CL2	1	float	4	3	ppm
	Ethylene Oxide	9385-P	SENSOR_GASES_PRO_ETO	14	ETO	1	float	4	3	ppm
	Hydrogen – H2	9380-P	SENSOR_GASES_PRO_H2	15	H2	1	float	4	3	ppm
	Hydrogen Sulphide – H2S	9381-P	SENSOR_GASES_PRO_H2S	16	H2S	1	float	4	3	ppm
	Hydrogen Chloride – HCL	9382-P	SENSOR_GASES_PRO_HCL	17	HCL	1	float	4	3	ppm
	Hydrogen Cyanide – HCN	9383-P	SENSOR_GASES_PRO_HCN	18	HCN	1	float	4	3	ppm
	Phospine – PH3	9384-P	SENSOR_GASES_PRO_PH3	19	PH3	1	float	4	3	ppm
	Sulfur Dioxide – SO2	9377-P	SENSOR_GASES_PRO_SO2	20	SO2	1	float	4	3	ppm
	P&S! SOCKET A (gas sensor)	N/A	SENSOR_GASES_PRO_SOCKET_A	30	GP_A	1	float	4	3	ppm
	P&S! SOCKET B (gas sensor)	N/A	SENSOR_GASES_PRO_SOCKET_B	31	GP_B	1	float	4	3	ppm
	P&S! SOCKET C (gas sensor)	N/A	SENSOR_GASES_PRO_SOCKET_C	32	GP_C	1	float	4	3	ppm
	P&S! SOCKET F (gas sensor)	N/A	SENSOR_GASES_PRO_SOCKET_F	35	GP_F	1	float	4	3	ppm
	Particle Matter – PM1	9387-P	SENSOR_GASES_PRO_PM1	70	PM1	1	float	4	4	μg/m3
	Particle matter – PM2.5	9387-P	SENSOR_GASES_PRO_PM2_5	71	PM2_5	1	float	4	4	μg/m3
	Particle Matter – PM10	9387-P	SENSOR_GASES_PRO_PM10	72	PM10	1	float	4	4	μg/m3
	Particle Matter – 24 bins	9387-P	SENSOR_GASES_PRO_PM_BIN	190	PM_BIN	24	uint16_t	2	0	Particles
	Particle Matter – First 16 bins	9387-P	SENSOR_GASES_PRO_PM_BINL	191	PM_BINL	16	uint16_t	2	0	Particles
	Particle Matter – Last 8 bins	9387-P	SENSOR_GASES_PRO_PM_BINH	192	PM_BINH	8	uint16_t	2	0	Particles
	BME – Temperature Celsius	9370-P	SENSOR_GASES_PRO_TC	74	TC	1	float	4	2	° C
	BME – Temperature Fahrenheit	9370-P	SENSOR_GASES_PRO_TF	75	TF	1	float	4	2	° F
	BME – Humidity	9370-P	SENSOR_GASES_PRO_HUM	76	HUM	1	float	4	1	%RH
	BME – Pressure	9370-P	SENSOR_GASES_PRO_PRES	77	PRES	1	float	4	2	Pascales
	Luxes	9325	SENSOR_GASES_PRO_LUXES	78	LUX	1	uint32_t	4	0	luxes
	Ultrasound	9246-P	SENSOR_GASES_PRO_US	79	US	1	uint16_t	2	0	cm

	Sensor	Sensor Reference	Sensor TAG	SENSOR ID		Number Of Fields	Binary		ASCII	Unit
				Binary	ASCII		Type of variable	Size per Field (Bytes)	Default Decimal Precision	
Events v30 (Smart Security)	Water flow	"9296 / 9297 / 9298"	SENSOR_EVENTS_WF	40	WF	1	float	4	3	l/min
	PIR	9212	SENSOR_EVENTS_PIR	41	PIR	1	uint8_t	1	0	Open / Closed
	Liquid presence	9243	SENSOR_EVENTS_LP	42	LP	1	uint8_t	1	0	Open / Closed
	Liquid level	"9239 / 9240 / 9242"	SENSOR_EVENTS_LL	43	LL	1	uint8_t	1	0	Open / Closed
	Hall effect	9207	SENSOR_EVENTS_HALL	44	HALL	1	uint8_t	1	0	Open / Closed
	Relay input	N/A	SENSOR_EVENTS_RELAY_IN	45	RIN	1	uint8_t	1	0	Open / Closed
	Relay output	N/A	SENSOR_EVENTS_RELAY_OUT	46	ROUT	1	uint8_t	1	0	Open / Closed
	P&S! SOCKET A (binary)	N/A	SENSOR_EVENTS_SOCKET_A	47	EV_A	1	uint8_t	1	0	Open / Closed
	P&S! SOCKET C (binary)	N/A	SENSOR_EVENTS_SOCKET_C	48	EV_C	1	uint8_t	1	0	Open / Closed
	P&S! SOCKET D (binary)	N/A	SENSOR_EVENTS_SOCKET_D	49	EV_D	1	uint8_t	1	0	Open / Closed
	P&S! SOCKET E (binary)	N/A	SENSOR_EVENTS_SOCKET_E	50	EV_E	1	uint8_t	1	0	Open / Closed
	BME – Temperature Celsius	9370-P	SENSOR_EVENTS_TC	74	TC	1	float	4	2	° C
	BME – Temperature Fahrenheit	9370-P	SENSOR_EVENTS_TF	75	TF	1	float	4	2	° F
	BME – Humidity	9370-P	SENSOR_EVENTS_HUM	76	HUM	1	float	4	1	%RH
	BME – Pressure	9370-P	SENSOR_EVENTS_PRES	77	PRES	1	float	4	2	Pascales
	Luxes	9325	SENSOR_EVENTS_LUXES	78	LUX	1	uint32_t	4	0	luxes
	Ultrasound	9246-P	SENSOR_EVENTS_US	79	US	1	uint16_t	2	0	cm

	Sensor	Sensor Reference	Sensor TAG	SENSOR ID		Number Of Fields	Binary		ASCII	Unit
				Binary	ASCII		Type of variable	Size per Field (Bytes)	Default Decimal Precision	
Smart Cities PRO v30	Carbon Monoxide – CO	9386-P	SENSOR_CITIES_PRO_CO	0	CO	1	float	4	3	ppm
	Carbon Dioxide – CO2	9371-P	SENSOR_CITIES_PRO_CO2	1	CO2	1	float	4	3	ppm
	Oxygen – O2	9385-P	SENSOR_CITIES_PRO_O2	2	O2	1	float	4	3	ppm
	Methane – CH4	9380-P	SENSOR_CITIES_PRO_CH4	3	CH4	1	float	4	3	% LEL
	Ozone – O3	9381-P	SENSOR_CITIES_PRO_O3	4	O3	1	float	4	3	ppm
	Ammonia – NH3	9382-P	SENSOR_CITIES_PRO_NH3	5	NH3	1	float	4	3	ppm
	Nitrogen Dioxide – NO2	9383-P	SENSOR_CITIES_PRO_NO2	6	NO2	1	float	4	3	ppm
	Nitrogen Monoxide – NO	9378-P	SENSOR_CITIES_PRO_NO	12	NO	1	float	4	3	ppm
	Chlorine – CL2	9375-P	SENSOR_CITIES_PRO_CL2	13	CL2	1	float	4	3	ppm
	Ethylene Oxide	9376-P	SENSOR_CITIES_PRO_ETO	14	ETO	1	float	4	3	ppm
	Hydrogen – H2	9373-P	SENSOR_CITIES_PRO_H2	15	H2	1	float	4	3	ppm
	Hydrogen Sulphide – H2S	9384-P	SENSOR_CITIES_PRO_H2S	16	H2S	1	float	4	3	ppm
	Hydrogen Chloride – HCL	9377-P	SENSOR_CITIES_PRO_HCL	17	HCL	1	float	4	3	ppm
	Hydrogen Cyanide – HCN	9379-P	SENSOR_CITIES_PRO_HCN	18	HCN	1	float	4	3	ppm
	Phospine – PH3	9374-P	SENSOR_CITIES_PRO_PH3	19	PH3	1	float	4	3	ppm
	Sulfur Dioxide – SO2	9372-P	SENSOR_CITIES_PRO_SO2	20	SO2	1	float	4	3	ppm
	Noise Level	TBD	SENSOR_CITIES_PRO_NOISE	21	NOISE	1	float	4	2	dBA
	P&S! SOCKET B (gas sensor)	N/A	SENSOR_CITIES_PRO_SOCKET_B	31	GP_B	1	float	4	3	ppm
	P&S! SOCKET C (gas sensor)	N/A	SENSOR_CITIES_PRO_SOCKET_C	32	GP_C	1	float	4	3	ppm
	P&S! SOCKET F (gas sensor)	N/A	SENSOR_CITIES_PRO_SOCKET_F	35	GP_F	1	float	4	3	ppm
	Dust sensor (PM1)	9387-P	SENSOR_CITIES_PRO_PM1	70	PM1	1	float	4	4	µg/m3
	Dust sensor (PM2.5)	9387-P	SENSOR_CITIES_PRO_PM2_5	71	PM2_5	1	float	4	4	µg/m3
	Dust sensor (PM10)	9387-P	SENSOR_CITIES_PRO_PM10	72	PM10	1	float	4	4	µg/m3
	BME – Temperature Celsius	9370-P	SENSOR_CITIES_PRO_TC	74	TC	1	float	4	2	° C
	BME – Temperature Fahrenheit	9370-P	SENSOR_CITIES_PRO_TF	75	TF	1	float	4	2	° F
	BME – Humidity	9370-P	SENSOR_CITIES_PRO_HUM	76	HUM	1	float	4	1	%RH
	BME – Pressure	9370-P	SENSOR_CITIES_PRO_PRES	77	PRES	1	float	4	2	Pascales
	Luxes	9325	SENSOR_CITIES_PRO_LUXES	78	LUX	1	uint32_t	4	0	luxes
	Ultrasound	9246-P	SENSOR_CITIES_PRO_US	79	US	1	uint16_t	2	0	cm

	Sensor	Sensor Reference	Sensor TAG	SENSOR ID		Number Of Fields	Binary		ASCII	Unit
				Binary	ASCII		Type of variable	Size per Field (Bytes)	Default Decimal Precision	
Smart Water Ions v30	Calcium Ions	9352 / 9414	SENSOR_IONS_CA	100	SWICA	1	float	4	3	ppm
	Fluoride Ions	9353 / 9417	SENSOR_IONS_FL	101	SWIFL	1	float	4	3	ppm
	Fluoroborate Ions	9354	SENSOR_IONS_FB	102	SWIFB	1	float	4	3	ppm
	Nitrate Ions	9355 / 9421	SENSOR_IONS_NO3	103	SWINO3	1	float	4	3	ppm
	Bromide Ions	9356/ 9413	SENSOR_IONS_BR	104	SWIBR	1	float	4	3	ppm
	Chloride Ions	9357 / 9415	SENSOR_IONS_CL	105	SWICL	1	float	4	3	ppm
	Cupric Ions	9358 / 9416	SENSOR_IONS_CU	106	SWICU	1	float	4	3	ppm
	Iodide Ions	9360 / 9418	SENSOR_IONS_IO	107	SWIIO	1	float	4	3	ppm
	Ammonium	9412	SENSOR_IONS_NH4	108	SWINH4	1	float	4	3	ppm
	Silver Ions	9362 / 9425	SENSOR_IONS_AG	109	SWIAG	1	float	4	3	ppm
	pH	9363 / 9411	SENSOR_IONS_PH	110	SWIPH	1	float	4	3	ppm
	Lithium Ions	9419	SENSOR_IONS_LI	111	SWILI	1	float	4	3	ppm
	Magnesium Ions	9420	SENSOR_IONS_MG	112	SWIMG	1	float	4	3	ppm
	Nitrite Ions	9422	SENSOR_IONS_NO2	113	SWINO2	1	float	4	3	ppm
	Perchlorate Ions	9423	SENSOR_IONS_CLO4	114	SWICLO4	1	float	4	3	ppm
	Potassium Ions	9424	SENSOR_IONS_K	115	SWIK	1	float	4	3	ppm
	Sodium Ions	9426	SENSOR_IONS_NA	116	SWINA	1	float	4	3	ppm
	P&S! SOCKET A (ions)	N/A	SENSOR_IONS_SOCKET_A	117	SWI_A	1	float	4	3	ppm
	P&S! SOCKET B (ions)	N/A	SENSOR_IONS_SOCKET_B	118	SWI_B	1	float	4	3	ppm
	P&S! SOCKET C (ions)	N/A	SENSOR_IONS_SOCKET_C	119	SWI_C	1	float	4	3	ppm
	P&S! SOCKET D (ions)	N/A	SENSOR_IONS_SOCKET_D	120	SWI_D	1	float	4	3	ppm
	Water Temperature	9255	SENSOR_IONS_WT	134	WT	1	float	4	2	ºC
Radiation	Geiger tube	N/A	SENSOR_RADIATION	129	RAD	1	float	4	6 or 0	uSv/h or cpm
Smart Water v30	Water pH	9328	SENSOR_WATER_PH	130	PH	1	float	4	2	N/A
	Oxidation Reduction Potential	9329	SENSOR_WATER_ORP	131	ORP	1	float	4	3	voltage
	Disolved oxygen	9327	SENSOR_WATER_DO	132	DO	1	float	4	1	%
	Water Conductivity	9326	SENSOR_WATER_COND	133	COND	1	float	4	1	μS/cm
	Water Temperature	9255	SENSOR_WATER_WT	134	WT	1	float	4	2	ºC
	Turbidity	9353	SENSOR_WATER_TURB	135	TURB	1	float	4	1	NTU
	pH (P&S! SOCKET A)	9328	SENSOR_WATER_PH_A	136	PH_A	1	float	4	2	N/A
	pH (P&S! SOCKET E)	9328	SENSOR_WATER_PH_E	137	PH_E	1	float	4	2	N/A
	ORP (P&S! SOCKET A)	9329	SENSOR_WATER_ORP_A	138	ORP_A	1	float	4	3	voltage
	ORP (P&S! SOCKET E)	9329	SENSOR_WATER_ORP_E	139	ORP_E	1	float	4	3	voltage

	Sensor	Sensor Reference	Sensor TAG	SENSOR ID		Number Of Fields	Binary		ASCII	Unit
				Binary	ASCII		Type of variable	Size per Field (Bytes)	Default Decimal Precision	
Smart Agriculture	Soil Moisture (watermark1)	9248	SENSOR_AGR_SOIL1	150	SOIL1	1	float	4	2	Frequency
	Soil Moisture (watermark2)	9248	SENSOR_AGR_SOIL2	151	SOIL2	1	float	4	2	Frequency
	Soil Moisture (watermark3)	9248	SENSOR_AGR_SOIL3	152	SOIL3	1	float	4	2	Frequency
	Soil Temperature (DS18B20/PT1000)	86949/9255	SENSOR_AGR_SOILTC	153	SOILTC	1	float	4	2	ºC
	Soil Temperature (DS18B20/PT1000)	86949/9255	SENSOR_AGR_SOILTF	154	SOILTF	1	float	4	2	ºF
	Leaf Wetness	9249	SENSOR_AGR_LW	155	LW	1	float	4	3	%
	Anemometer	9256	SENSOR_AGR_ANE	156	ANE	1	float	4	2	km/h
	Wind Vane	9256	SENSOR_AGR_WV	157	WV	1	uint8_t	1	N/A	Direction
	Pluviometer (current hour)	9256	SENSOR_AGR_PLV1	158	PLV1	1	float	4	2	mm
	Pluviometer (previous hour)	9256	SENSOR_AGR_PLV2	159	PLV2	1	float	4	2	mm/h
	Pluviometer (last 24h)	9256	SENSOR_AGR_PLV3	160	PLV3	1	float	4	2	mm/day
	Solar Radiation	9251	SENSOR_AGR_PAR	161	PAR	1	float	4	2	μmol*m-2*s-1
	Ultraviolet Radiation	9257	SENSOR_AGR_UV	162	UV	1	float	4	2	μmol*m-2*s-1
	Trunk Diameter	9252	SENSOR_AGR_TD	163	TD	1	float	4	3	mm
	Stem Diameter	9253	SENSOR_AGR_SD	164	SD	1	float	4	3	mm
	Fruit Diameter	9254	SENSOR_AGR_FD	165	FD	1	float	4	3	mm
	Soil Moisture (P&S! SOCKET B)	9248	SENSOR_AGR_SOIL_B	166	SOIL_B	1	float	4	2	Frequency
	Soil Moisture (P&S! SOCKET C)	9248	SENSOR_AGR_SOIL_C	167	SOIL_C	1	float	4	2	Frequency
	Soil Moisture (P&S! SOCKET E)	9248	SENSOR_AGR_SOIL_E	168	SOIL_E	1	float	4	2	Frequency
	BME – Temperature Celsius	9370-P	SENSOR_AGR_TC	74	TC	1	float	4	2	° C
	BME – Temperature Fahrenheit	9370-P	SENSOR_AGR_TF	75	TF	1	float	4	2	° F
	BME – Humidity	9370-P	SENSOR_AGR_HUM	76	HUM	1	float	4	1	%RH
	BME – Pressure	9370-P	SENSOR_AGR_PRES	77	PRES	1	float	4	2	Pascales
	Luxes	9325	SENSOR_AGR_LUXES	78	LUX	1	uint32_t	4	0	luxes
	Ultrasound	9246-P	SENSOR_AGR_US	79	US	1	uint16_t	2	0	cm
Ambient Control	BME – Temperature Celsius	9370-P	SENSOR_AMBIENT_TC	74	TC	1	float	4	2	° C
	BME – Temperature Fahrenheit	9370-P	SENSOR_AMBIENT_TF	75	TF	1	float	4	2	° F
	BME - Humidity	9370-P	SENSOR_AMBIENT_HUM	76	HUM	1	float	4	1	%RH
	BME - Pressure	9370-P	SENSOR_AMBIENT_PRES	77	PRES	1	float	4	2	Pascales
	Luminosity	9205	SENSOR_AMBIENT_LUM	172	LUM	1	float	4	3	Ohms
	Luxes	9325	SENSOR_AMBIENT_LUXES	78	LUX	1	uint32_t	4	0	luxes

	Sensor	Sensor Reference	Sensor TAG	SENSOR ID		Number Of Fields	Binary		ASCII	Unit
				Binary	ASCII		Type of variable	Size per Field (Bytes)	Default Decimal Precision	
Additional	Battery level	N/A	SENSOR_BAT	52	BAT	1	uint8_t	1	0	%
	Global Positioning System	N/A	SENSOR_GPS	53	GPS	2	float	4	6	degrees
	RSSI	N/A	SENSOR_RSSI	54	RSSI	1	int	2	0	N/A
	MAC Address	N/A	SENSOR_MAC	55	MAC	1	string	variable	N/A	N/A
	Network Address (XBee)	N/A	SENSOR_NA	56	NA	1	string	variable	N/A	N/A
	Network ID origin (XBee)	N/A	SENSOR_NID	57	NID	1	string	variable	N/A	N/A
	Date	N/A	SENSOR_DATE	58	DATE	3	uint8_t	1	N/A	N/A
	Time	N/A	SENSOR_TIME	59	TIME	3	uint8_t	1	N/A	N/A
	GMT	N/A	SENSOR_GMT	60	GMT	1	int	1	N/A	N/A
	Free_RAM	N/A	SENSOR_RAM	61	RAM	1	int	2	0	bytes
	Internal_temperature	N/A	SENSOR_IN_TEMP	62	IN_TEMP	1	float	4	2	ºC
	Accelerometer	N/A	SENSOR_ACC	63	ACC	3	int	2	0	mg
	Millis	N/A	SENSOR_MILLIS	64	MILLIS	1	uint32_t	4	0	ms
	String	N/A	SENSOR_STR	65	STR	1	string	variable	N/A	N/A
	Unique Identifier	N/A	SENSOR_UID	68	UID	1	string	variable	N/A	N/A
	RFID block	N/A	SENSOR_RB	69	RB	1	string	variable	N/A	N/A
	Dust sensor (PM1)	9387-P	SENSOR_DUST_PM1	70	PM1	1	float	4	4	µg/m3
	Dust sensor (PM2.5)	9387-P	SENSOR_DUST_PM2_5	71	PM2_5	1	float	4	4	µg/m3
	Dust sensor (PM10)	9387-P	SENSOR_DUST_PM10	72	PM10	1	float	4	4	µg/m3
	BME – Temperature Celsius	9370-P	SENSOR_BME_TC	74	TC	1	float	4	2	° C
	BME – Temperature Fahrenheit	9370-P	SENSOR_BME_TF	75	TF	1	float	4	2	° F
	BME – Humidity	9370-P	SENSOR_BME_HUM	76	HUM	1	float	4	1	%RH
	BME – Pressure	9370-P	SENSOR_BME_PRES	77	PRES	1	float	4	2	Pascales
	Luxes	9325	SENSOR_LUXES	78	LUX	1	uint32_t	4	0	luxes
	Ultrasound	9246-P	SENSOR_ULTRASOUND	79	US	1	uint16_t	2	0	cm
	GPS speed over the ground	N/A	SENSOR_SPEED	89	SPEED_OG	1	float	4	2	km/h
	GPS course over the ground	N/A	SENSOR_COURSE	90	COURSE_OG	1	float	4	2	degrees
	GPS altitude	N/A	SENSOR_ALTITUDE	91	ALT	1	float	4	2	m
	GPS HDOP	N/A	SENSOR_HDOP	92	HDOP	1	float	4	3	N/A
	GPS VDOP	N/A	SENSOR_VDOP	93	VDOP	1	float	4	3	N/A
	GPS PDOP	N/A	SENSOR_PDOP	94	PDOP	1	float	4	3	N/A
	Timestamp (Unix/Epoch)	N/A	SENSOR_TST	123	TST	1	uint32_t	4	0	seconds
	Version of API	N/A	SENSOR_VAPI	125	VAPI	1	uint8_t	1	N/A	N/A
	Version of program	N/A	SENSOR_VPROG	126	VPROG	1	uint8_t	1	N/A	N/A
	Version of bootloader	N/A	SENSOR_VBOOT	127	VBOOT	1	uint8_t	1	N/A	N/A
	Parking state	N/A	SENSOR_PARKING	128	PS	1	uint8_t	1	N/A	N/A

	Sensor	Sensor Reference	Sensor TAG	SENSOR ID		Number Of Fields	Binary		ASCII	Unit
				Binary	ASCII		Type of variable	Size per Field (Bytes)	Default Decimal Precision	
4-20 mA Current Loop	**Current(P&S! SOCKET A)**	N/A	SENSOR_4_20_CURRENT_SOCKET_A	175	CUR_A	1	float	4	3	mA
	Current(P&S! SOCKET B)	N/A	SENSOR_4_20_CURRENT_SOCKET_B	176	CUR_B	1	float	4	3	mA
	Current(P&S! SOCKET C)	N/A	SENSOR_4_20_CURRENT_SOCKET_C	177	CUR_C	1	float	4	3	mA
	Current(P&S! SOCKET D)	N/A	SENSOR_4_20_CURRENT_SOCKET_D	178	CUR_D	1	float	4	3	mA
Industrial Protocols	Modbus coils(up to 16 bits)	N/A	SENSOR_MODBUS_COILS	180	MB_COILS	2	int	2	N/A	N/A
	Modbus discrete inputs (up to 16 bits)	N/A	SENSOR_MODBUS_DISCRETE_INPUT	181	MB_DI	2	int	2	N/A	N/A
	Modbus holding registers (up to 2 registers)	N/A	SENSOR_MODBUS_HOLDING_REGS	182	MB_HR	3	int	2	N/A	N/A
	Modbus input registers (up to 2 registers)	N/A	SENSOR_MODBUS_INPUT_REGS	183	MB_IR	3	int	2	N/A	N/A
	CAN bus engine rpm	N/A	SENSOR_CANBUS_RPM	184	CB_RPM	1	unit16_t	2	0	rpm
	CAN bus vehicle speed	N/A	SENSOR_CANBUS_VS	185	CB_VS	1	unit16_t	2	0	km/h
	CAN bus fuel rate	N/A	SENSOR_CANBUS_FR	186	CB_FR	1	unit16_t	2	0	l/h
	CAN bus fuel level	N/A	SENSOR_CANBUS_FL	187	CB_FL	1	unit8_t	1	0	%
	CAN bus throttle position	N/A	SENSOR_CANBUS_TP	188	CB_TP	1	unit8_t	1	0	%
	CAN bus fuel pressure	N/A	SENSOR_CANBUS_FP	189	CB_FP	1	unit16_t	2	0	kPa

Figure: Field types

	Sensor	Sensor Reference	Sensor TAG	SENSOR ID		Number Of Fields	Binary		ASCII	Units
				Binary	ASCII		Type of variable	Size Field (Bytes)	Default Decimal Precision	
Smart Agriculture Xtreme	BME – Temperature Celsius socket A	9370-P	AGRX_TC_A	0	TC_A	1	float	4	2	ºC
	BME – Temperature Fahrenheit socket A	9370-P	AGRX_TF_A	1	TF_A	1	float	4	2	ºF
	BME – Humidity socket A	9370-P	AGRX_HUM_A	2	HUM_A	1	float	4	1	%RH
	BME- Pressure socket A	9370-P	AGRX_PRES_A	3	PRES_A	1	float	4	2	Pascales
	BME – Temperature Celsius socket D	9370-P	AGRX_TC_D	4	TC_D	1	float	4	2	ºC
	BME – Temperature Fahrenheit socket D	9370-P	AGRX_TF_D	5	TF_D	1	float	4	2	ºF
	BME – Humidity socket D	9370-P	AGRX_HUM_D	6	HUM_D	1	float	4	1	%RH
	BME- Pressure socket D	9370-P	AGRX_PRES_D	7	PRES_D	1	float	4	2	Pascales
	Luxes socket A	9325-P	AGRX_LUXES_A	8	LUX_A	1	uint32_t	4	0	Luxes
	Luxes socket D	9325-P	AGRX_LUXES_D	9	LUX_D	1	uint32_t	4	0	Luxes
	Ultrasound socket A	9246-P	AGRX_US_A	10	US_A	1	uint16_t	2	0	cm
	Ultrasound socket D	9246-P	AGRX_US_D	11	US_D	1	uint16_t	2	0	cm
	Leaf wetness	9466-P	AGRX_LW	12	LW	1	float	4	4	V
	Shortwave radiation socket B	9470-P	AGRX_SR_B	13	SR_B	1	float	4	2	µmol*m2*s-1
	Shortwave radiation socket C	9470-P	AGRX_SR_C	14	SR_C	1	float	4	2	µmol*m2*s-1
	Shortwave radiation socket E	9470-P	AGRX_SR_E	15	SR_E	1	float	4	2	µmol*m2*s-1
	Shortwave radiation socket F	9470-P	AGRX_SR_F	16	SR_F	1	float	4	2	µmol*m2*s-1
	PAR – socket B	9251-P	AGRX_PAR_B	17	PAR_B	1	float	4	2	µmol*m2*s-1
	PAR – socket C	9251-P	AGRX_PAR_C	18	PAR_C	1	float	4	2	µmol*m2*s-1
	PAR – socket E	9251-P	AGRX_PAR_E	19	PAR_E	1	float	4	2	µmol*m2*s-1
	PAR – socket F	9251-P	AGRX_PAR_F	20	PAR_F	1	float	4	2	µmol*m2*s-1
	Ultraviolet radiation – socket B	9257-P	AGRX_UV_B	21	UV_B	1	float	4	2	µmol*m2*s-1
	Ultraviolet radiation – socket C	9257-P	AGRX_UV_C	22	UV_C	1	float	4	2	µmol*m2*s-1
	Ultraviolet radiation – socket E	9257-P	AGRX_UV_E	23	UV_E	1	float	4	2	µmol*m2*s-1
	Ultraviolet radiation – socket F	9257-P	AGRX_UV_F	24	UV_F	1	float	4	2	µmol*m2*s-1
	Trunk Diameter	9252-P	AGRX_TD	25	TD	1	float	4	3	mm
	Stem Diameter	9253-P	AGRX_SD	26	SD	1	float	4	3	mm
	Fruit Diameter	9254-P	AGRX_FD	27	FD	1	float	4	3	mm
	Current socket F	N/A	AGRX_CURRENT_SOCKET_F	28	CUR_F	1	float	4	3	mA
	Current socket B	N/A	AGRX_CURRENT_SOCKET_B	29	CUR_B	1	float	4	3	mA
	SI-411 – target temperature socket A	9468-P	AGRX_SI411_TC1_A	30	TC1_A	1	float	4	4	ºC
	SI-411 – target temperature socket B	9468-P	AGRX_SI411_TC1_B	31	TC1_B	1	float	4	4	ºC

Smart Agriculture Xtreme	SI-411 – target temperature socket C	9468-P	AGRX_SI411_TC1_C	32	TC1_C	1	float	4	4	ºC
	SI-411 – target temperature socket D	9468-P	AGRX_SI411_TC1_D	33	TC1_D	1	float	4	4	ºC
	SI-411 – millivolts socket A	9468-P	AGRX_SI411_MV1_A	34	MV1_A	1	float	4	3	mV
	SI-411 – millivolts socket B	9468-P	AGRX_SI411_MV1_B	35	MV1_B	1	float	4	3	mV
	SI-411 – millivolts socket C	9468-P	AGRX_SI411_MV1_C	36	MV1_C	1	float	4	3	mV
	SI-411 – millivolts socket D	9468-P	AGRX_SI411_MV1_D	37	MV1_D	1	float	4	3	mV
	SI-411 – body temperature socket A	9468-P	AGRX_SI411_BT1_A	38	BT1_A	1	float	4	3	ºC
	SI-411 – body temperature socket B	9468-P	AGRX_SI411_BT1_B	39	BT1_B	1	float	4	3	ºC
	SI-411 – body temperature socket C	9468-P	AGRX_SI411_BT1_C	40	BT1_C	1	float	4	3	ºC
	SI-411 – body temperature socket D	9468-P	AGRX_SI411_BT1_D	41	BT1_D	1	float	4	3	ºC
	Modbus coils (up to 16 bits)	N/A	AGRX_MODBUS_COILS	42	MB_COILS	2	int	2	N/A	N/A
	Modbus discrete inputs (up to 16 bits)	N/A	AGRX_MODBUS_DISCRETE_INPUT	43	MB_DI	2	int	2	N/A	N/A
	Modbus holding registers (up to 2 registers)	N/A	AGRX_MODBUS_HOLDING_REGS	44	MB_HR	3	int	2	N/A	N/A
	Modbus input registers (up to 2 registers)	N/A	AGRX_MODBUS_INPUT_REGS	45	MB_IR	3	int	2	N/A	N/A
	CAN bus engine rpm	N/A	AGRX_CANBUS_RPM	46	CB_RPM	1	uint16_t	2	0	rpm
	CAN bus vehicle speed	N/A	AGRX_CANBUS_VS	47	CB_VS	1	uint16_t	2	0	km/h
	CAN bus fuel rate	N/A	AGRX_CANBUS_FR	48	CB_FR	1	uint16_t	2	0	l/h
	CAN bus fuel level	N/A	AGRX_CANBUS_FL	49	CB_FL	1	uint8_t	1	0	%
	CAN bus throttle position	N/A	AGRX_CANBUS_TP	50	CB_TP	1	uint8_t	1	0	%
	CAN bus fuel pressure	N/A	AGRX_CANBUS_FP	51	CB_FP	1	uint16_t	2	0	kPa
	Datasol Met - radiation		Datasol Met – radiation	95	RAD	1	unit16_t	2	0	W/m^{2}
	Datasol Met – semicell 1 radiation		AGRX_DATASOL_SC1_RAD	96	SC1_RAD	1	uint16_t	2	0	W/m²
	Datasol Met – semicell 2 radiation		AGRX_DATASOL_SC2_RAD	97	SC2_RAD	1	uint16_t	2	0	W/m²
	Datasol Met – environment temperature		AGRX_DATASOL_ETC	98	ETC	1	float	4	1	ºC
	Datasol Met – panel temperature		AGRX_DATASOL_PTC	99	PTC	1	float	4	1	ºC
	Datasol Met – wind speed		AGRX_DATASOL_WSP	100	WSP	1	float	4	1	m/s
	Datasol Met – peak sun hours		AGRX_DATASOL_PSH	101	PSH	1	float	4	2	h
	Datasol Met – necessary cleaning notice		AGRX_DATASOL_NCN	102	NCN	1	unit8_t	1	0	Y/N
	SO-411 – calibrated oxigen socket A	9469-P	AGRX_SO411_CO_A	134	CO_A	1	float	4	3	%
	SO-411 – calibrated oxigen socket B	9469-P	AGRX_SO411_CO_B	135	CO_B	1	float	4	3	%
	SO-411 – calibrated oxigen socket C	9469-P	AGRX_SO411_CO_C	136	CO_C	1	float	4	3	%
	SO-411 – calibrated oxigen socket D	9469-P	AGRX_SO411_CO_D	137	CO_D	1	float	4	3	%
	SO-411 – body temperature socket A	9469-P	AGRX_SO411_TC2_A	138	TC2_A	1	float	4	1	ºC
	SO-411 – body temperature socket B	9469-P	AGRX_SO411_TC2_B	139	TC2_B	1	float	4	1	ºC
	SO-411 – body temperature socket C	9469-P	AGRX_SO411_TC2_C	140	TC2_C	1	float	4	1	ºC
	SO-411 – body temperature socket D	9469-P	AGRX_SO411_TC2_D	141	TC2_D	1	float	4	1	ºC
	SO-411 – millivolts socket A	9469-P	AGRX_SO411_MV2_A	142	MV2_A	1	float	4	4	mV

Smart Agriculture Xtreme	SO-411 – millivolts socket B	9469-P	AGRX_SO411_MV2_B	143	MV2_B	1	float	4	4	mV
	SO-411 – millivolts socket C	9469-P	AGRX_SO411_MV2_C	144	MV2_C	1	float	4	4	mV
	SO-411 – millivolts socket D	9469-P	AGRX_SO411_MV2_D	145	MV2_D	1	float	4	4	mV
	GS3 – dielectric permitivity socket A	9464-P	AGRX_GS3_DP1_A	146	DP1_A	1	float	4	2	N/A
	GS3 – dielectric permitivity socket B	9464-P	AGRX_GS3_DP1_B	147	DP1_B	1	float	4	2	N/A
	GS3 – dielectric permitivity socket C	9464-P	AGRX_GS3_DP1_C	148	DP1_C	1	float	4	2	N/A
	GS3 – dielectric permitivity socket D	9464-P	AGRX_GS3_DP1_D	149	DP1_D	1	float	4	2	N/A
	GS3 – electrical conductivity socket A	9464-P	AGRX_GS3_EC1_A	150	EC1_A	1	float	4	0	µS/cm
	GS3 – electrical conductivity socket B	9464-P	AGRX_GS3_EC1_B	151	EC1_B	1	float	4	0	µS/cm
	GS3 – electrical conductivity socket D	9464-P	AGRX_GS3_EC1_D	153	EC1_D	1	float	4	0	µS/cm
	GS3 – temperature socket A	9464-P	AGRX_GS3_TC3_A	154	TC3_A	1	float	4	2	ºC
	GS3 – temperature socket B	9464-P	AGRX_GS3_TC3_B	155	TC3_B	1	float	4	2	ºC
	GS3 – temperature socket C	9464-P	AGRX_GS3_TC3_C	156	TC3_C	1	float	4	2	ºC
	GS3 – temperature socket D	9464-P	AGRX_GS3_TC3_D	157	TC3_D	1	float	4	2	ºC
	5TE – dielectric permitivity socket A	9402-P	AGRX_5TE_DP2_A	158	DP2_A	1	float	4	2	N/A
	5TE – dielectric permitivity socket B	9402-P	AGRX_5TE_DP2_B	159	DP2_B	1	float	4	2	N/A
	5TE – dielectric permitivity socket C	9402-P	AGRX_5TE_DP2_C	160	DP2_C	1	float	4	2	N/A
	5TE – dielectric permitivity socket D	9402-P	AGRX_5TE_DP2_D	161	DP2_D	1	float	4	2	N/A
	5TE – electrical conductivity socket A	9402-P	AGRX_5TE_EC2_A	162	EC2_A	1	float	4	2	dS/m
	5TE – electrical conductivity socket B	9402-P	AGRX_5TE_EC2_B	163	EC2_B	1	float	4	2	dS/m
	5TE – electrical conductivity socket C	9402-P	AGRX_5TE_EC2_C	164	EC2_C	1	float	4	2	dS/m
	5TE – electrical conductivity socket D	9402-P	AGRX_5TE_EC2_D	165	EC2_D	1	float	4	2	dS/m
	5TE – temperature socket A	9402-P	AGRX_5TE_TC4_A	166	TC4_A	1	float	4	1	ºC
	5TE – temperature socket B	9402-P	AGRX_5TE_TC4_B	167	TC4_B	1	float	4	1	ºC
	5TE – temperature socket C	9402-P	AGRX_5TE_TC4_C	168	TC4_C	1	float	4	1	ºC
	5TE – temperature socket D	9402-P	AGRX_5TE_TC4_D	169	TC4_D	1	float	4	1	ºC
	VP4 – vapor pressure socket A	9471-P	AGRX_VP4_VP_A	170	VP_A	1	float	4	3	kPa
	VP4 – vapor pressure socket B	9471-P	AGRX_VP4_VP_B	171	VP_B	1	float	4	3	kPa
	VP4 – vapor pressure socket C	9471-P	AGRX_VP4_VP_C	172	VP_C	1	float	4	3	kPa
	VP4 – vapor pressure socket D	9471-P	AGRX_VP4_VP_D	173	VP_D	1	float	4	3	kPa
	VP4 – temperature socket A	9471-P	AGRX_VP4_TC5_A	174	TC5_A	1	float	4	1	ºC
	VP4 – temperature socket B	9471-P	AGRX_VP4_TC5_B	175	TC5_B	1	float	4	1	ºC
	VP4 – temperature socket C	9471-P	AGRX_VP4_TC5_C	176	TC5_C	1	float	4	1	ºC
	VP4 – temperature socket D	9471-P	AGRX_VP4_TC5_D	177	TC5_D	1	float	4	1	ºC
	VP4 – relative humidity socket A	9471-P	AGRX_VP4_RH_A	178	RH_A	1	float	4	1	%RH
	VP4 – relative humidity socket B	9471-P	AGRX_VP4_RH_B	179	RH_B	1	float	4	1	%RH
	VP4 – relative humidity socket C	9471-P	AGRX_VP4_RH_C	180	RH_C	1	float	4	1	%RH
	VP4 – relative humidity socket D	9471-P	AGRX_VP4_RH_D	181	RH_D	1	float	4	1	%RH

Smart Agriculture Xtreme	VP4 – Atmospheric pressure socket A	9471-P	AGRX_VP4_AP_A	182	AP_A	1	float	4	2	kPa
	VP4 – Atmospheric pressure socket B	9471-P	AGRX_VP4_AP_B	183	AP_B	1	float	4	2	kPa
	VP4 – Atmospheric pressure socket C	9471-P	AGRX_VP4_AP_C	184	AP_C	1	float	4	2	kPa
	VP4 – Atmospheric pressure socket D	9471-P	AGRX_VP4_AP_D	185	AP_D	1	float	4	2	kPa
	MPS6 – water potential socket A	9465-P	AGRX_MPS6_WP_A	186	WP_A	1	float	4	1	kPa
	MPS6 – water potential socket B	9465-P	AGRX_MPS6_WP_B	187	WP_B	1	float	4	1	kPa
	MPS6 – water potential socket C	9465-P	AGRX_MPS6_WP_C	188	WP_C	1	float	4	1	kPa
	MPS6 – water potential socket D	9465-P	AGRX_MPS6_WP_D	189	WP_D	1	float	4	1	kPa
	MPS6 – temperature socket A	9465-P	AGRX_MPS6_TC6_A	190	TC6_A	1	float	4	1	ºC
	MPS6 – temperature socket B	9465-P	AGRX_MPS6_TC6_B	191	TC6_B	1	float	4	1	ºC
	MPS6 – temperature socket C	9465-P	AGRX_MPS6_TC6_C	192	TC6_C	1	float	4	1	ºC
	MPS6 – temperature socket D	9465-P	AGRX_MPS6_TC6_D	193	TC6_D	1	float	4	1	ºC
	SF421 – bud temperature socket A	9467-P	AGRX_SF421_BT2_A	194	BT2_A	1	float	4	3	ºC
	SF421 – bud temperature socket B	9467-P	AGRX_SF421_BT2_B	195	BT2_B	1	float	4	3	ºC
	SF421 – bud temperature socket C	9467-P	AGRX_SF421_BT2_C	196	BT2_C	1	float	4	3	ºC
	SF421 – bud temperature socket D	9467-P	AGRX_SF421_BT2_D	197	BT2_D	1	float	4	3	ºC
	SF421 – leaf temperature socket A	9467-P	AGRX_SF421_LT_A	198	LT_A	1	float	4	3	ºC
	SF421 – leaf temperature socket B	9467-P	AGRX_SF421_LT_B	199	LT_B	1	float	4	3	ºC
	SF421 – leaf temperature socket C	9467-P	AGRX_SF421_LT_C	200	LT_C	1	float	4	3	ºC
	SF421 – leaf temperature socket D	9467-P	AGRX_SF421_LT_D	201	LT_D	1	float	4	3	ºC
	5TM – dielectric permitivity socket A	9460-P	AGRX_5TM_DP3_A	202	DP3_A	1	float	4	2	?
	5TM – dielectric permitivity socket B	9460-P	AGRX_5TM_DP3_B	203	DP3_B	1	float	4	2	?
	5TM – dielectric permitivity socket C	9460-P	AGRX_5TM_DP3_C	204	DP3_C	1	float	4	2	?
	5TM – dielectric permitivity socket D	9460-P	AGRX_5TM_DP3_D	205	DP3_D	1	float	4	2	?
	5TM – temperature socket A	9460-P	AGRX_5TM_TC7_A	206	TC7_A	1	float	4	1	ºC
	5TM – temperature socket B	9460-P	AGRX_5TM_TC7_B	207	TC7_B	1	float	4	1	ºC
	5TM – temperature socket C	9460-P	AGRX_5TM_TC7_C	208	TC7_C	1	float	4	1	ºC
	5TM – temperature socket D	9460-P	AGRX_5TM_TC7_D	209	TC7_D	1	float	4	1	ºC
	GMXXXX – wind direction	N/A	AGRX_GMX_WD	210	WD	1	uint16_t	2	0	º
	GMXXXX – average wind direction	N/A	AGRX_GMX_AWD	211	AWD	1	uint16_t	2	0	º
	GMXXXX – wind speed	N/A	AGRX_GMX_WS	212	WS	1	float	4	2	m/s
	GMXXXX – average wind speed	N/A	AGRX_GMX_AWS	213	AWS	1	float	4	2	m/s
	GMXXXX – average_wind_gust_ direction	N/A	AGRX_GMX_AWGD	214	AWGD	1	uint16_t	2	0	º
	GMXXXX – average_wind_gust_speed	N/A	AGRX_GMX_AWGS	215	AWGS	1	float	4	2	m/s
	GMXXXX – wind sensor status	N/A	AGRX_GMX_WSS	216	WSS	1	string	4	N/A	N/A
	GMXXXX – precipitation total	N/A	AGRX_GMX_PT	217	PT	1	float	4	3	mm
	GMXXXX – precipitation intensity	N/A	AGRX_GMX_PI	218	PI	1	float	4	3	mm
	GMXXXX – precipitation status	N/A	AGRX_GMX_PST	219	PST	1	uint8_t	1	0	Y/N

Smart Agricutlure Xtreme	GMXXXX – corrected wind direction	N/A	AGRX_GMX_CWD	220	CWD	1	uint16_t	2	0	°
	GMXXXX – average corrected wind direction	N/A	AGRX_GMX_ACWD	221	ACWD	1	uint16_t	2	0	°
	GMXXXX – compass	N/A	AGRX_GMX_CMPS	222	CMPS	1	uint16_t	2	0	°
	GMXXXX – x tilt	N/A	AGRX_GMX_XT	223	XT	1	float	4	0	°
	GMXXXX – y tilt	N/A	AGRX_GMX_YT	224	YT	1	float	4	0	°
	GMXXXX – z orient	N/A	AGRX_GMX_ZO	225	ZO	1	float	4	0	1/-1
	GMXXXX – supply voltage	N/A	AGRX_GMX_SVO	226	SVO	1	float	4	1	V
	GMXXXX – status	N/A	AGRX_GMX_ST	227	ST	1	string	4	N/A	N/A
	GMXXXX – Solar radiation	N/A	AGRX_GMX_SR	228	SR_WS	1	uint16_t	2	0	W/m^2
	GMXXXX – Solar sunshine hours	N/A	AGRX_GMX_SUNSHINE	229	SSH_WS	1	float	4	2	hours
	GMXXXX – sunrise	N/A	AGRX_GMX_SUNRISE	230	SRT_WS	1	string	5	0	h:min
	GMXXXX – solar noon	N/A	AGRX_GMX_SOLAR_NOON	231	SNT_WS	1	string	5	0	h:min
	GMXXXX – sunset	N/A	AGRX_GMX_SUNSET	232	ST_WS	1	string	5	0	h:min
	GMXXXX – position of the sun	N/A	AGRX_GMX_POS_SUN	233	PS_WS	1	string	7	0	°:°
	GMXXXX – twilight civil	N/A	AGRX_GMX_TW_CIV	234	TC_WS	1	string	5	0	h:min
	GMXXXX – twilight nautical	N/A	AGRX_GMX_TW_NAU	235	TN_WS	1	string	5	0	h:min
	GMXXXX – twilight astronomical	N/A	AGRX_GMX_TW_AST	236	TA_WS	1	string	5	0	h:min
	GMXXXX – Barometric pressure	N/A	AGRX_GMX_PRES	237	PRES_WS	1	float	4	1	hPa
	GMXXXX – Pressure at sea level	N/A	AGRX_GMX_PRES_SEA	238	PRESSL_WS	1	float	4	1	hPa
	GMXXXX – Pressure at station	N/A	AGRX_GMX_PRES_STA	239	PRESS_WS	1	float	4	1	hPa
	GMXXXX – Relative humidity	N/A	AGRX_GMX_RH	240	RH_WS	1	uint16_t	2	0	%
	GMXXXX – Air temperature	N/A	AGRX_GMX_TEMP	241	TEM_WS	1	float	4	1	°C
	GMXXXX – dewpoint	N/A	AGRX_GMX_DEWP	242	DP_WS	1	float	4	1	°C
	GMXXXX – Absolute Humidity	N/A	AGRX_GMX_AH	243	AH_WS	1	float	4	2	gm-3
	GMXXXX – Air density	N/A	AGRX_GMX_AD	244	AD_WS	1	float	4	1	Kgm-3
	GMXXXX – Wet Bulb temperature	N/A	AGRX_GMX_WBT	245	WBT_WS	1	float	4	1	°C
	GMXXXX – Wind chill	N/A	AGRX_GMX_WC	246	WC_WS	1	float	4	0	°C
	GMXXXX – Heat Index	N/A	AGRX_GMX_HI	247	HI_WS	1	uint16_t	2	0	°C
	GMXXXX – Corrected Speed	N/A	AGRX_GMX_GPS_CSP	248	GPS_CSP	1	float	4	2	m/s
	GMXXXX – Average corrected Speed	N/A	AGRX_GMX_GPS_ACS	249	GPS_ACSP	1	float	4	2	m/s
	GMXXXX – Corrected Gust speed	N/A	AGRX_GMX_GPS_CGS	250	GPS_GSP	1	float	4	2	m/s
	GMXXXX – Corrected gust direction	N/A	AGRX_GMX_GPS_CGD	251	GPS_GDIR	1	uint16_t	2	0	°
	GMXXXX – GPS location (lat / long)	N/A	AGRX_GMX_GPS_LOC	252	GPS_LOC	1	string	28	0	°:°:m
	GMXXXX – GPS heading	N/A	AGRX_GMX_GPS_HEA	253	GPS_H	1	uint16_t	2	0	°
	GMXXXX – Speed	N/A	AGRX_GMX_GPS_SPEED	254	GPS_SP	1	float	4	2	m/s
	GMXXXX – GPS status	N/A	AGRX_GMX_GPS_STATUS	255	GPS_ST	1	string	4	0	N/A

Figure: Field types for Smart Agriculture Xtreme

	Sensor	Sensor Reference	Sensor TAG	SENSOR ID		Number Of Fields	Binary		ASCII	Units
				Binary	ASCII		Type of variable	Size Field (Bytes)	Default Decimal Precision	
Smart Water Xtreme	BME – Temperature Celsius socket A	9370-P	WTRX_TC_A	0	TC_A	1	float	4	2	ºC
	BME – Temperature Fahrenheit socket A	9370-P	WTRX_TF_A	1	TF_A	1	float	4	2	ºF
	BME – Humidity socket A	9370-P	WTRX_HUM_A	2	HUM_A	1	float	4	1	%RH
	BME- Pressure socket A	9370-P	WTRX_PRES_A	3	PRES_A	1	float	4	2	Pascales
	BME – Temperature Celsius socket D	9370-P	WTRX_TC_D	4	TC_D	1	float	4	2	ºC
	BME – Temperature Fahrenheit socket D	9370-P	WTRX_TF_D	5	TF_D	1	float	4	2	ºF
	BME – Humidity socket D	9370-P	WTRX_HUM_D	6	HUM_D	1	float	4	1	%RH
	BME- Pressure socket D	9370-P	WTRX_PRES_D	7	PRES_D	1	float	4	2	Pascales
	Luxes socket A	9325-P	WTRX_LUXES_A	8	LUX_A	1	uint32_t	4	0	Luxes
	Luxes socket D	9325-P	WTRX_LUXES_D	9	LUX_D	1	uint32_t	4	0	Luxes
	Ultrasound socket A	9246-P	WTRX_US_A	10	US_A	1	uint16_t	2	0	cm
	Ultrasound socket D	9246-P	WTRX_US_D	11	US_D	1	uint16_t	2	0	cm
	OPTOD – Temperature socket A	9488-P	WTRX_OPTOD_TC1_A	12	TC1_A	1	float	4	2	ºC
	OPTOD – Temperature socket B	9488-P	WTRX_OPTOD_TC1_B	13	TC1_B	1	float	4	2	ºC
	OPTOD – Temperature socket C	9488-P	WTRX_OPTOD_TC1_C	14	TC1_C	1	float	4	2	ºC
	OPTOD – Temperature socket D	9488-P	WTRX_OPTOD_TC1_D	15	TC1_D	1	float	4	2	ºC
	OPTOD – Temperature socket E	9488-P	WTRX_OPTOD_TC1_E	16	TC1_E	1	float	4	2	ºC
	OPTOD – Oxygen saturation socket A	9488-P	WTRX_OPTOD_OS_A	17	OS_A	1	float	4	2	%
	OPTOD – Oxygen saturation socket B	9488-P	WTRX_OPTOD_OS_B	18	OS_B	1	float	4	2	%
	OPTOD – Oxygen saturation socket C	9488-P	WTRX_OPTOD_OS_C	19	OS_C	1	float	4	2	%
	OPTOD – Oxygen saturation socket D	9488-P	WTRX_OPTOD_OS_D	20	OS_D	1	float	4	2	%
	OPTOD – Oxygen saturation socket E	9488-P	WTRX_OPTOD_OS_E	21	OS_E	1	float	4	2	%
	OPTOD – Oxygen MGL socket A	9488-P	WTRX_OPTOD_OM_A	22	OM_A	1	float	4	2	mg/L
	OPTOD – Oxygen MGL socket B	9488-P	WTRX_OPTOD_OM_B	23	OM_B	1	float	4	2	mg/L
	OPTOD – Oxygen MGL socket C	9488-P	WTRX_OPTOD_OM_C	24	OM_C	1	float	4	2	mg/L
	OPTOD – Oxygen MGL socket D	9488-P	WTRX_OPTOD_OM_D	25	OM_D	1	float	4	2	mg/L
	OPTOD – Oxygen MGL socket E	9488-P	WTRX_OPTOD_OM_E	26	OM_E	1	float	4	2	mg/L
	OPTOD – Oxygen PPM socket A	9488-P	WTRX_OPTOD_OP_A	27	OP_A	1	float	4	2	ppm
	OPTOD – Oxygen PPM socket B	9488-P	WTRX_OPTOD_OP_B	28	OP_B	1	float	4	2	ppm
	OPTOD – Oxygen PPM socket C	9488-P	WTRX_OPTOD_OP_C	29	OP_C	1	float	4	2	ppm
	OPTOD – Oxygen PPM socket D	9488-P	WTRX_OPTOD_OP_D	30	OP_D	1	float	4	2	ppm
	OPTOD – Oxygen PPM socket E	9488-P	WTRX_OPTOD_OP_E	31	OP_E	1	float	4	2	ppm

Smart Water Xtreme	Modbus coils (up to 16 bits)	N/A	WTRX_MODBUS_COILS	42	MB_COILS	2	int	2	N/A	N/A
	Modbus discrete inputs (up to 16 bits)	N/A	WTRX_MODBUS_DISCRETE_INPUT	43	MB_DI	2	int	2	N/A	N/A
	Modbus holding registers (up to 2 registers)	N/A	WTRX_MODBUS_HOLDING_REGS	44	MB_HR	3	int	2	N/A	N/A
	Modbus input registers (up to 2 registers)	N/A	WTRX_MODBUS_INPUT_REGS	45	MB_IR	3	int	2	N/A	N/A
	CAN bus engine rpm	N/A	WTRX_CANBUS_RPM	46	CB_RPM	1	uint16_t	2	0	rpm
	CAN bus vehicle speed	N/A	WTRX_CANBUS_VS	47	CB_VS	1	uint16_t	2	0	km/h
	CAN bus fuel rate	N/A	WTRX_CANBUS_FR	48	CB_FR	1	uint16_t	2	0	l/h
	CAN bus fuel level	N/A	WTRX_CANBUS_FL	49	CB_FL	1	uint8_t	1	0	%
	CAN bus throttle position	N/A	WTRX_CANBUS_TP	50	CB_TP	1	uint8_t	1	0	%
	CAN bus fuel pressure	N/A	WTRX_CANBUS_FP	51	CB_FP	1	uint16_t	2	0	kPa
	PHEHT – Temperature socket A	9485-P	WTRX_PHEHT_TC2_A	134	TC2_A	1	float	4	2	ºC
	PHEHT – Temperature socket B	9485-P	WTRX_PHEHT_TC2_B	135	TC2_B	1	float	4	2	ºC
	PHEHT – Temperature socket C	9485-P	WTRX_PHEHT_TC2_C	136	TC2_C	1	float	4	2	ºC
	PHEHT – Temperature socket D	9485-P	WTRX_PHEHT_TC2_D	137	TC2_D	1	float	4	2	ºC
	PHEHT – Temperature socket E	9485-P	WTRX_PHEHT_TC2_E	138	TC2_E	1	float	4	2	ºC
	PHEHT – pH socket A	9485-P	WTRX_PHEHT_PH_A	139	PH_A	1	float	4	2	pH
	PHEHT – pH socket B	9485-P	WTRX_PHEHT_PH_B	140	PH_B	1	float	4	2	pH
	PHEHT – pH socket C	9485-P	WTRX_PHEHT_PH_C	141	PH_C	1	float	4	2	pH
	PHEHT – pH socket D	9485-P	WTRX_PHEHT_PH_D	142	PH_D	1	float	4	2	pH
	PHEHT – pH socket E	9485-P	WTRX_PHEHT_PH_E	143	PH_E	1	float	4	2	pH
	PHEHT – Redox socket A	9485-P	WTRX_PHEHT_RX_A	144	RX_A	1	float	4	2	mV
	PHEHT – Redox socket B	9485-P	WTRX_PHEHT_RX_B	145	RX_B	1	float	4	2	mV
	PHEHT – Redox socket C	9485-P	WTRX_PHEHT_RX_C	146	RX_C	1	float	4	2	mV
	PHEHT – Redox socket D	9485-P	WTRX_PHEHT_RX_D	147	RX_D	1	float	4	2	mV
	PHEHT – Redox socket E	9485-P	WTRX_PHEHT_RX_E	148	RX_E	1	float	4	2	mV
	PHEHT – pHMV socket A	9485-P	WTRX_PHEHT_PM_A	149	PM_A	1	float	4	2	mV
	PHEHT – pHMV socket B	9485-P	WTRX_PHEHT_PM_B	150	PM_B	1	float	4	2	mV
	PHEHT – pHMV socket C	9485-P	WTRX_PHEHT_PM_C	151	PM_C	1	float	4	2	mV
	PHEHT – pHMV socket D	9485-P	WTRX_PHEHT_PM_D	152	PM_D	1	float	4	2	mV
	PHEHT – pHMV socket E	9485-P	WTRX_PHEHT_PM_E	153	PM_E	1	float	4	2	mV
	C4E – Temperature socket A	9486-P	WTRX_C4E_TC3_A	154	TC3_A	1	float	4	2	ºC
	C4E – Temperature socket B	9486-P	WTRX_C4E_TC3_B	155	TC3_B	1	float	4	2	ºC
	C4E – Temperature socket C	9486-P	WTRX_C4E_TC3_C	156	TC3_C	1	float	4	2	ºC
	C4E – Temperature socket D	9486-P	WTRX_C4E_TC3_D	157	TC3_D	1	float	4	2	ºC
	C4E – Temperature socket E	9486-P	WTRX_C4E_TC3_E	158	TC3_E	1	float	4	2	ºC
	C4E – Conductivity socket A	9486-P	WTRX_C4E_CN_A	159	CN_A	1	float	4	2	μS/cm
	C4E – Conductivity socket B	9486-P	WTRX_C4E_CN_B	160	CN_B	1	float	4	2	μS/cm
	C4E – Conductivity socket C	9486-P	WTRX_C4E_CN_C	161	CN_C	1	float	4	2	μS/cm

Smart Water Xtreme	C4E – Conductivity socket D	9486-P	WTRX_C4E_CN_D	162	CN_D	1	float	4	2	μS/cm
	C4E – Conductivity socket E	9486-P	WTRX_C4E_CN_E	163	CN_E	1	float	4	2	μS/cm
	C4E – Salinity socket A	9486-P	WTRX_C4E_SA_A	164	SA_A	1	float	4	2	ppt
	C4E – Salinity socket B	9486-P	WTRX_C4E_SA_B	165	SA_B	1	float	4	2	ppt
	C4E – Salinity socket C	9486-P	WTRX_C4E_SA_C	166	SA_C	1	float	4	2	ppt
	C4E – Salinity socket D	9486-P	WTRX_C4E_SA_D	167	SA_D	1	float	4	2	ppt
	C4E – Salinity socket E	9486-P	WTRX_C4E_SA_E	168	SA_E	1	float	4	2	ppt
	C4E – Total dissolved solids socket A	9486-P	WTRX_C4E_TD_A	169	TD_A	1	float	4	2	ppm
	C4E – Total dissolved solids socket B	9486-P	WTRX_C4E_TD_B	170	TD_B	1	float	4	2	ppm
	C4E – Total dissolved solids socket C	9486-P	WTRX_C4E_TD_C	171	TD_C	1	float	4	2	ppm
	C4E – Total dissolved solids socket D	9486-P	WTRX_C4E_TD_D	172	TD_D	1	float	4	2	ppm
	C4E – Total dissolved solids socket E	9486-P	WTRX_C4E_TD_E	173	TD_E	1	float	4	2	ppm
	NTU – Temperature socket A	9353-PX	WTRX_NTU_TC4_A	174	TC4_A	1	float	4	2	ºC
	NTU – Temperature socket B	9353-PX	WTRX_NTU_TC4_B	175	TC4_B	1	float	4	2	ºC
	NTU – Temperature socket C	9353-PX	WTRX_NTU_TC4_C	176	TC4_C	1	float	4	2	ºC
	NTU – Temperature socket D	9353-PX	WTRX_NTU_TC4_D	177	TC4_D	1	float	4	2	ºC
	NTU – Temperature socket E	9353-PX	WTRX_NTU_TC4_E	178	TC4_E	1	float	4	2	ºC
	NTU – Turbidity NTU socket A	9353-PX	WTRX_NTU_TN_A	179	TN_A	1	float	4	2	NTU
	NTU – Turbidity NTU socket B	9353-PX	WTRX_NTU_TN_B	180	TN_B	1	float	4	2	NTU
	NTU – Turbidity NTU socket C	9353-PX	WTRX_NTU_TN_C	181	TN_C	1	float	4	2	NTU
	NTU – Turbidity NTU socket D	9353-PX	WTRX_NTU_TN_D	182	TN_D	1	float	4	2	NTU
	NTU – Turbidity NTU socket E	9353-PX	WTRX_NTU_TN_E	183	TN_E	1	float	4	2	NTU
	NTU – Turbidity MGL socket A	9353-PX	WTRX_NTU_TM_A	184	TM_A	1	float	4	2	mg/l
	NTU – Turbidity MGL socket B	9353-PX	WTRX_NTU_TM_B	185	TM_B	1	float	4	2	mg/l
	NTU – Turbidity MGL socket C	9353-PX	WTRX_NTU_TM_C	186	TM_C	1	float	4	2	mg/l
	NTU – Turbidity MGL socket D	9353-PX	WTRX_NTU_TM_D	187	TM_D	1	float	4	2	mg/l
	NTU – Turbidity MGL socket E	9353-PX	WTRX_NTU_TM_E	188	TM_E	1	float	4	2	mg/l
	CTZN – Temperature socket A	9487-P	WTRX_CTZN_TC5_A	189	TC5_A	1	float	4	2	ºC
	CTZN – Temperature socket B	9487-P	WTRX_CTZN_TC5_B	190	TC5_B	1	float	4	2	ºC
	CTZN – Temperature socket C	9487-P	WTRX_CTZN_TC5_C	191	TC5_C	1	float	4	2	ºC
	CTZN – Temperature socket D	9487-P	WTRX_CTZN_TC5_D	192	TC5_D	1	float	4	2	ºC
	CTZN – Temperature socket E	9487-P	WTRX_CTZN_TC5_E	193	TC5_E	1	float	4	2	ºC
	CTZN – Conductivity socket A	9487-P	WTRX_CTZN_CN1_A	194	CN1_A	1	float	4	2	mS/cm
	CTZN – Conductivity socket B	9487-P	WTRX_CTZN_CN1_B	195	CN1_B	1	float	4	2	mS/cm
	CTZN – Conductivity socket C	9487-P	WTRX_CTZN_CN1_C	196	CN1_C	1	float	4	2	mS/cm
	CTZN – Conductivity socket D	9487-P	WTRX_CTZN_CN1_D	197	CN1_D	1	float	4	2	mS/cm
	CTZN – Conductivity socket E	9487-P	WTRX_CTZN_CN1_E	198	CN1_E	1	float	4	2	mS/cm
	CTZN – Salinity socket A	9487-P	WTRX_CTZN_SA1_A	199	SA1_A	1	float	4	2	ppt

Smart Water Xtreme	CTZN – Salinity socket B	9487-P	WTRX_CTZN_SA1_B	200	SA1_B	1	float	4	2	ppt
	CTZN – Salinity socket C	9487-P	WTRX_CTZN_SA1_C	201	SA1_C	1	float	4	2	ppt
	CTZN – Salinity socket D	9487-P	WTRX_CTZN_SA1_D	202	SA1_D	1	float	4	2	ppt
	CTZN – Salinity socket_E	9487-P	WTRX_CTZN_SA1_E	203	SA1_E	1	float	4	2	ppt
	CTZN – Conductivity not compensated socket A	9487-P	WTRX_CTZN_CU_A	204	CU_A	1	float	4	2	mS/cm
	CTZN – Conductivity not compensated socket B	9487-P	WTRX_CTZN_CU_B	205	CU_B	1	float	4	2	mS/cm
	CTZN – Conductivity not compensated socket C	9487-P	WTRX_CTZN_CU_C	206	CU_C	1	float	4	2	mS/cm
	CTZN – Conductivity not compensated socket D	9487-P	WTRX_CTZN_CU_D	207	CU_D	1	float	4	2	mS/cm
	CTZN – Conductivity not compensated socket E	9487-P	WTRX_CTZN_CU_E	208	CU_E	1	float	4	2	mS/cm
	MES5 – Temperature socket A	9490-P	WTRX_MES5_TC6_A	209	TC6_A	1	float	4	2	ºC
	MES5 – Temperature socket B	9490-P	WTRX_MES5_TC6_B	210	TC6_B	1	float	4	2	ºC
	MES5 – Temperature socket C	9490-P	WTRX_MES5_TC6_C	211	TC6_C	1	float	4	2	ºC
	MES5 – Temperature socket D	9490-P	WTRX_MES5_TC6_D	212	TC6_D	1	float	4	2	ºC
	MES5 – Temperature socket E	9490-P	WTRX_MES5_TC6_E	213	TC6_E	1	float	4	2	ºC
	MES5 – Sludge Blanket socket A	9490-P	WTRX_MES5_SB_A	214	SB_A	1	float	4	2	%
	MES5 – Sludge Blanket socket B	9490-P	WTRX_MES5_SB_B	215	SB_B	1	float	4	2	%
	MES5 – Sludge Blanket socket C	9490-P	WTRX_MES5_SB_C	216	SB_C	1	float	4	2	%
	MES5 – Sludge Blanket socket D	9490-P	WTRX_MES5_SB_D	217	SB_D	1	float	4	2	%
	MES5 – Sludge Blanket socket E	9490-P	WTRX_MES5_SB_E	218	SB_E	1	float	4	2	%
	MES5 – Suspended solids socket A	9490-P	WTRX_MES5_SS_A	219	SS_A	1	float	4	2	g/L
	MES5 – Suspended solids socket B	9490-P	WTRX_MES5_SS_B	220	SS_B	1	float	4	2	g/L
	MES5 – Suspended solids socket C	9490-P	WTRX_MES5_SS_C	221	SS_C	1	float	4	2	g/L
	MES5 – Suspended solids socket D	9490-P	WTRX_MES5_SS_D	222	SS_D	1	float	4	2	g/L
	MES5 – Suspended solids socket E	9490-P	WTRX_MES5_SS_E	223	SS_E	1	float	4	2	g/L
	MES5 – Turbidity FAU socket A	9490-P	WTRX_MES5_TF_A	224	TFA_A	1	float	4	2	FAU
	MES5 – Turbidity FAU socket B	9490-P	WTRX_MES5_TF_B	225	TFA_B	1	float	4	2	FAU
	MES5 – Turbidity FAU socket C	9490-P	WTRX_MES5_TF_C	226	TFA_C	1	float	4	2	FAU
	MES5 – Turbidity FAU socket D	9490-P	WTRX_MES5_TF_D	227	TFA_D	1	float	4	2	FAU
	MES5 – Turbidity FAU socket E	9490-P	WTRX_MES5_TF_E	228	TFA_E	1	float	4	2	FAU

Figure: Field types for Smart Water Xtreme

3. Frame Functions

The following sections show how to create frames and add sensor fields.

3.1. Setting the Waspmote Identifier

The `setID()` function allows the user to store the Waspmote ID in the EEPROM memory. The Waspmote ID will be used in order to set the corresponding field in the frame's header when calling the `createFrame()` function.

Example of use:

```
{
  // store Waspmote ID in EEPROM memory (16-byte max)
  frame.setID("Waspmote_Pro");
}
```

3.2. Creating new frames

The `createFrame()` function creates a new frame structure. It clears the frame buffer and inserts the header information: start delimiter, frame type, Serial ID, Waspmote ID and sequence number. After calling this function, the `addSensor()` function should be used in order to insert sensor fields within the payload. The function parameter selects the frame type:

- **ASCII**
- **BINARY**

Besides, it is possible to define the Waspmote ID which will be included in the frame's header (16 bytes maximum) instead of using the mote identifier stored in the EEPROM memory.

The function prototypes are the following:

- Create an ASCII frame. The Waspmote ID is taken from the EEPROM memory that `setID()` function has previously set:

```
{
	frame.createFrame(ASCII);
}
```

- Create an ASCII frame. The Waspmote ID (i.e. "Waspmote_Pro") is set as an input parameter:

```
{
	frame.createFrame(ASCII,"Waspmote_Pro");
}
```

- Create a Binary frame. The Waspmote ID (i.e. "Waspmote_Pro") is set as an input parameter:

```
{
	frame.createFrame(BINARY,"Waspmote_Pro");

}
```

3.3. Setting the frame size

The class constructor initializes the attribute `_maxSize` to `MAX_FRAME` constant, which is used to limit the maximum frame size. This constant defines a maximum **default size of 255 bytes** per frame. As this is the maximum possible value, it can be modified in WaspFrame.h in order to create frames with larger sizes.

On the other hand, `setFrameSize()` is the function which permits to set the frame size according to the user's consideration. Besides, it is possible to set the frame size depending on the XBee module, link encryption mode and AES encryption use. The following table defines the maximum frame size to be used for each communication protocol and several encryption possibilities:

Module			Maximum frame size
XBee-PRO 802.15.4	Link Encrypted	@16bit Unicast	98 bytes
		@64bit Unicast	94 bytes
		Broadcast	95 bytes
	Link Unencrypted		100 bytes
XBee 868LP			255 bytes
XBee-PRO 900HP			255 bytes
XBee-PRO DigiMesh			73 bytes
XBee-PRO ZigBee	Link Encrypted	@64bit Unicast	66 bytes
		Broadcast	84 bytes
	Link Unencrypted	@64bit Unicast	74 bytes
		Broadcast	92 bytes
Bluetooth – transparent connection			Limited by `MAX_FRAME`
GPRS / GPRS+GPS			Limited by `MAX_FRAME`
3G			Limited by `MAX_FRAME`
LoRa / SX1272 (868/900)			Limited by `MAX_FRAME`
WiFi PRO			Limited to a maximum 255 bytes for the complete command request
4G			Depends on the protocol used and `MAX_FRAME`

Figure: Maximum frame size per protocol

Note: `MAX_FRAME` *is 255 bytes but can be changed by the user.*

In the case that AES Encryption libraries are used to encrypt a Waspmote Frame, it is necessary to use the `setFrameSize()`. This function sets the maximum payload for the Waspmote Frame depending on the XBee protocol, addressing mode and link-encryption mode used.

The function prototypes are:

Set frame size depending on the protocol, addressing, link-encryption and encryption libraries used:

```
void setFrameSize(uint8_t protocol,
                  uint8_t addressing,
                  uint8_t linkEncryption,
                  uint8_t AESEncryption);
```

Where "protocol" specifies the XBee module protocol between:

```
XBEE_802_15_4
ZIGBEE
DIGIMESH
XBEE_900HP
XBEE_868LP
```

"addressing" specifies the addressing mode between:

- `UNICAST_16B:` for Unicast 16-bit addressing (only for XBee-802.15.4)
- `UNICAST_64B:` for Unicast 64-bit addressing
- `BROADCAST_MODE:` for Broadcast addressing

"linkEncryption" specifies the XBee encryption mode between:

- `ENABLED` : 1
- `DISABLED` : 0

"AESEncryption" specifies if AES encryption is used or not:

- `ENABLED` : 1
- `DISABLED` : 0

Set frame size depending on the protocol and encryption used (default UNICAST_64B addressing):

```
void setFrameSize(uint8_t protocol,
                  uint8_t linkEncryption,
                  uint8_t AESEncryption);
```

Examples of use:

```
{
 // set frame size to 125 bytes
 frame.setFrameSize(125);

 // XBee-802, unicast 16-b addressing, XBee encryption Disabled, AES encryption Disabled
 frame.setFrameSize(XBEE_802_15_4, UNICAST_16B, DISABLED, DISABLED);

 // XBee-868, unicast 64-b addressing, XBee encryption Enabled, AES encryption Enabled
  frame.setFrameSize(XBEE_868, ENABLED, ENABLED);

  // XBee-ZigBee, Broadcast addressing, XBee encryption Enabled, AES encryption Disabled
  frame.setFrameSize(ZIGBEE, BROADCAST, ENABLED, DISABLED);

 // XBee-900, unicast 64-b addressing, XBee encryption Disabled, AES encryption Enabled
 frame.setFrameSize(XBEE_900, DISABLED, ENABLED);

 // XBee-Digimesh, Broadcast addressing, XBee encryption Enabled, AES encryption Enabled
 frame.setFrameSize(DIGIMESH, BROADCAST, ENABLED, ENABLED);

}
```

Set frame size via parameter given by the user:

```
void setFrameSize(uint8_t size);
```

Where "size" must be less than MAX_FRAME, if not MAX_FRAME will be set as frame maximum size

Example:

How to set the frame size depending on the protocol and encryption used:
http://www.libelium.com/development/waspmote/examples/frame-05-set-frame-size

3.4. Setting the frame type

There is a function which allows the user to set the required frame type. This function must be called after calling createFrame() function. In the case it is not called, a default information frame type is chosen by createFrame(). The function that permits the setting of the frame type is setFrameType(). It is possible to select between different constants predefined in WaspFrame.h in order to set the sort of packet to be sent:

- TIMEOUT_FRAME
- EVENT_FRAME
- ALARM_FRAME
- SERVICE1_FRAME
- SERVICE2_FRAME

These constants permit to set the Frame Type in spite of the frame mode (ascii or binary).

Example of use:

```
{
  frame.setFrameType(TIMEOUT_FRAME); // set a TIMEOUT frame type
}
```

Example:

How to set the frame type:
http://www.libelium.com/development/waspmote/examples/frame-06-set-frame-type

Note: Currently, this feature is not supported by Meshlium. Only default frame types are used (information and encrypted frames).

3.5. Adding sensor fields

The addSensor() function allows the user to append new sensor fields to the frame. The first parameter is the sensor tag to identify the sensor to be added (this is described in the “Sensor fields” section). The sensor identifier is followed up by the sensor values, which might be presented in various types: int, float, strings, etc. This function is defined by several prototypes so as to permit several possibilities. This function returns the current length of the frame. Each call to this function appends a new field if there is enough space for the new field. If this function attempts to insert a sensor field which exceeds the maximum frame buffer size, then the sensor field is not added and the function returns -1.

Depending on the sensor field a specific type is needed for Binary frames (described in the “Sensor fields” section). If a mismatch occurs, a message will appear through USB port. The sensor table shows the needed data type for each sensor.

Example of use:

```
{
  // set frame fields (String - char*)
  frame.addSensor(SENSOR_STR, (char*) “STRING”);

  // set frame fields (Battery sensor - uint8_t)
  frame.addSensor(SENSOR_BAT, (uint8_t) PWR.getBatteryLevel());
}
```

The last example would create a **frame payload** with the following structure (depending on the frame mode):

- **ASCII frame.** Payload length: 32 bytes

Payload																	
S	T	R	:	S	T	R	I	N	G	#	B	A	T	:	8	7	#
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Figure: ASCII frame payload example

- **Binary frame.** Payload length: 15 bytes

Payload				
SENSOR_ STR	Length	“STRING”	SENSOR_BAT	0x57
0	1	2-7	8	9

Figure: Binary frame payload example

Examples:

Create ASCII frames with simple sensor data (1 field per sensor):
http://www.libelium.com/development/waspmote/examples/frame-01-ascii-simple

Create ASCII frames with complex sensor data (more than 1 field per sensor):
http://www.libelium.com/development/waspmote/examples/frame-02-ascii-multiple

Create Binary frames with simple sensor data (1 field per sensor):
http://www.libelium.com/development/waspmote/examples/frame-03-binary-simple

Create Binary frames with complex sensor data (more than 1 field per sensor):
http://www.libelium.com/development/waspmote/examples/frame-04-binary-multiple

3.6. Adding new sensor types

In case the user is interested in adding new sensor types, this guide explains how to do this process.

a) Define the new sensor identifier. As the rest of the sensors, it is necessary to define a unique identifier for the new sensor in WaspFrameConstantsv15.h:

```
#define SENSOR_GASES_CO      0
#define SENSOR_GASES_CO2     1
#define SENSOR_GASES_O2      2
#define SENSOR_GASES_CH4     3
...
#define NEW_SENSOR           ?
```

b) Define label for the new sensor. As the rest of the sensors, it is necessary to define a unique label for the new sensor in WaspFrameConstantsv15.h:

```
prog_char     str_frame_00[]      PROGMEM = “CO”;      // 0
prog_char     str_frame_01[]      PROGMEM = “CO2”;     // 1
prog_char     str_frame_02[]      PROGMEM = “O2”;      // 2
prog_char     str_frame_03[]      PROGMEM = “CH4”;     // 3
...
prog_char     str_NEW[]     PROGMEM = “NEW_LABEL”;     // ?
```

c) Fill the Flash Memory tables respecting the defined index in section “a”. The Flash Memory tables are:

- FRAME_SENSOR_TABLE: This is a string table in order to define the sensor labels. For ASCII frames.
- FRAME_SENSOR_TYPE_TABLE: This is a uint8_t table which specifies the type of sensor depending on the type of value the user must put as input. Only for Binary frames.
- FRAME_SENSOR_FIELD_TABLE: This is a uint8_t table which specifies the number of fields for each sensor.
- FRAME_DECIMAL_TABLE: This is a uint8_t table which specifies the number of decimals a float must be set when adding each sensor to an ASCII frame.

3.7. Showing the current Frame

There is a function called `showFrame()` which prints the frame structure at the moment this function is called.

Example of use:

```
{
  frame.showFrame();
}
```

3.8. Frame fragmentation (only binary frames)

When using binary frames, it is possible to fragment original frames into different fragments that share the same header. For that purpose, the `createFragmentHeader()` function allows the user to initialize a frame fragmentation. This function creates the fragment header needed for all fragments to be generated in the rest of the fragmentation process. This function needs an input parameter related to the maximum fragment size (including all bytes: header and payload).

This feature is useful in order to split large binary frames into fragments when the communication modules does not permit payloads large enough (for example, Sigfox and LoRaWAN impose severe restrictions). This function returns the number of pending sensor fields to be attached to a fragment. Thus, unless this function returns ‘0’, it means there are still pending fields to be inserted and more fragment generation is needed.

The `generateFragment()` function allows the user to generate a new fragment from the original binary frame. Prior calling this function, the user must call the `createFragmentHeader()` function to make sure that there are pending fields to be inserted in a new fragment. This function also returns the number of pending fields.

The resulting fragments are stored in `bufferFragment` and `lengthFragment` attributes. So the user can access this data in order to send it via the communication module.

Imagine we create an original binary frame with ‘n’ sensor fields. We need to fragment the frames due to a maximum payload limitation. After fragmenting the original frame we will come up with two fragments as you can see in the next figures:

HEADER							PAYLOAD			
<=>	Frame Type	Num of bytes	Serial ID	Waspmote ID	#	Sequence	Sensor_1	Sensor_2	...	Sensor_n

Figure: Original binary frame structure

HEADER							PAYLOAD			
<=>	Frame Type	Num of bytes	Serial ID	Waspmote ID	#	Sequence	Sensor_1	Sensor_2	...	Sensor_m

Figure: Fragment 1

HEADER							PAYLOAD			
<=>	Frame Type	Num of bytes	Serial ID	Waspmote ID	#	Sequence	Sensor_m+1	Sensor_m+2	...	Sensor_n

Figure: Fragment 2

Example of use:

```
{
  frame.createFrame(BINARY);
  frame.addSensor(SENSOR_STR, “field 1”);
  frame.addSensor(SENSOR_STR, “field 2”);
  ... // add numerous sensor fields into the original frame
  frame.addSensor(SENSOR_STR, “field N”);

  // proceed to create fragments from the original binary frame
  // ‘80’ is the maximum fragment size
  while (frame.createFragmentHeader(80) > 0)
  {
        frame.generateFragment();
        USB.println(F(“Fragment frame:”));
        USB.printHexln(frame.bufferFragment, frame.lengthFragment);
  }
}
```

Complete example:

www.libelium.com/development/waspmote/examples/frame-08-fragment-frames

4. Encrypted frames

In this chapter we explain how to create encrypted frames using the Waspmote AES libraries.

4.1. Encrypted frame format

The encrypted frame is a special binary frame which encapsulates the real encrypted frame as the payload of a new frame. The format of the encrypted frame is as follows:

<=>	Frame Type	Num bytes	Serial ID	Encrypted Payload
3B	1B	1B	8B	Multiple of 16B

Figure: Encrypted frame format

Where the “Encrypted Payload” is the original frame after being encrypted using the AES algorithm.

Note: Regarding the old Waspmote v12 platform version, a different format is used. Please refer to the corresponding documentation

4.2. Device to gateway encryption

Every single node in the network has its own private key (ensuring authenticity). Also, the gateway of the network (Meshlium) contains all nodes’ keys in order to decrypt the packets received from the nodes. The information goes encrypted from each node to the gateway so the intermediate nodes cannot access to it.

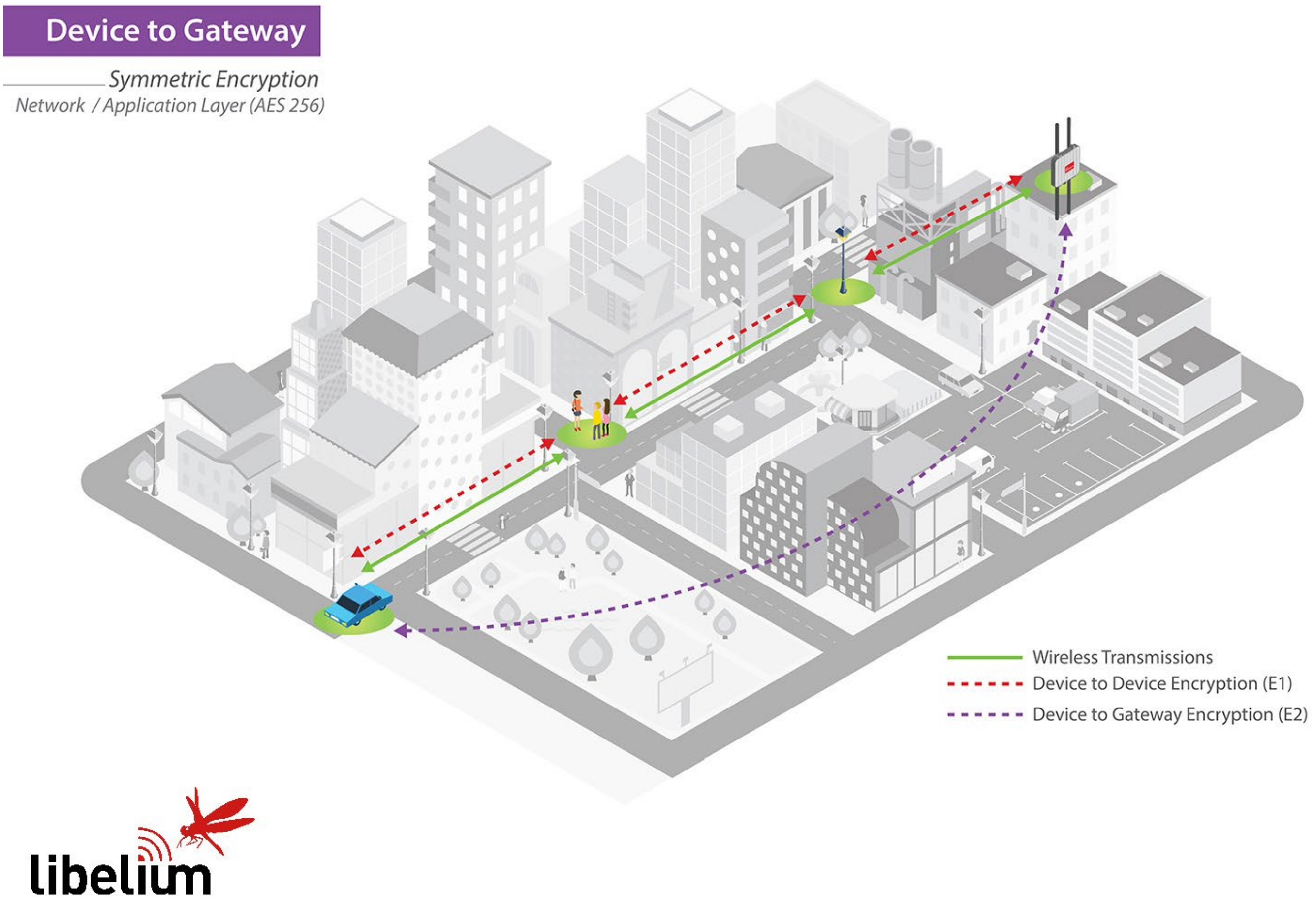

Device to Gateway
Symmetric Encryption
Network / Application Layer (AES 256)
Wireless Transmissions
Device to Device Encryption (E1)
Device to Gateway Encryption (E2)
libelium

Figure: Device to gateway encryption.

The process follows these steps:

1. Step 1: Create a new Frame (ASCII or BINARY)
2. Step 2: Encrypt the frame and use it as Encrypted Payload of the new Encrypted Frame

The function `encryptFrame()` is used to encrypt the original frame and generate the new one. It is necessary to indicate the type of AES encryption used regarding the key size: `AES_128`, `AES_192` or `AES_256`. Besides, the AES private key must be specified as a string of ASCII characters.

Example of use:

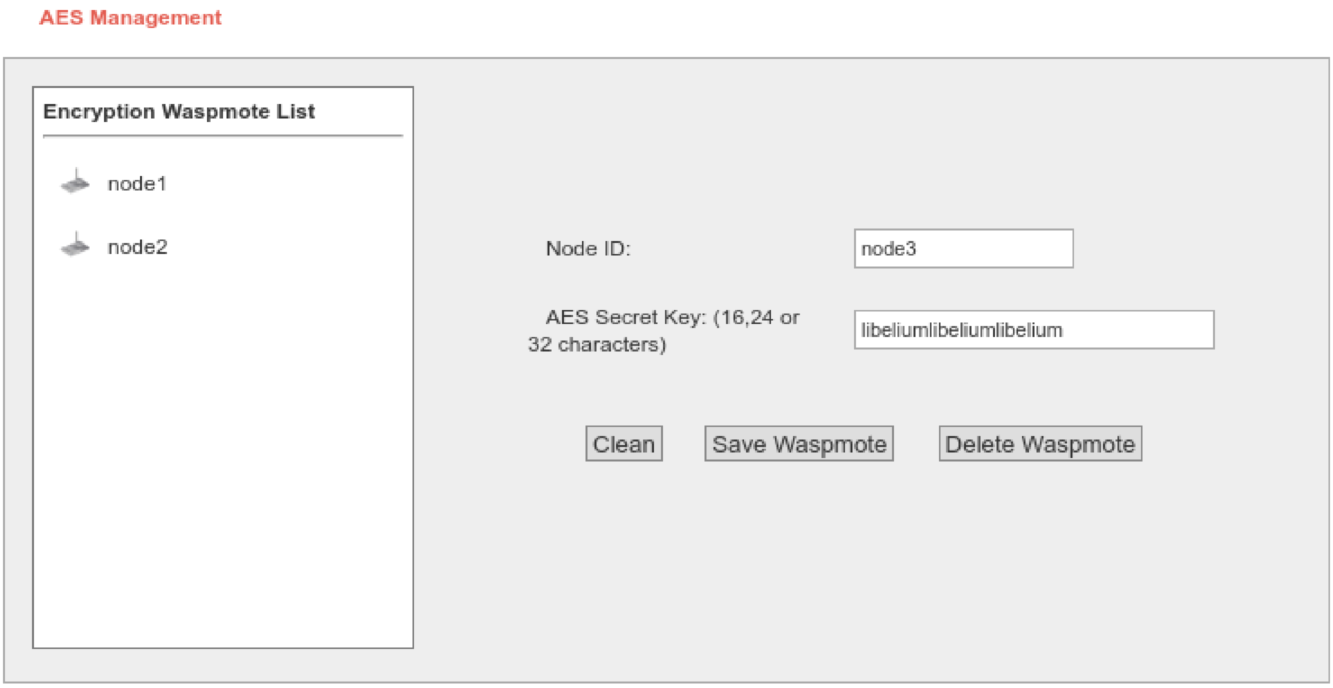
```
{
   // 1. create the original frame
   frame.createFrame( ASCII );
   frame.addSensor( SENSOR_BAT, battery_level );

   // 2. create the AES-128 encrypted frame
   frame.encryptFrame( AES_128, “libeliumlibelium” );

   // 3. show Encrypted frame contained in ‘frame.buffer’
   frame.showFrame();
}
```

Example of how to use encryption with Waspmote Frames:
www.libelium.com/development/waspmote/examples/frame-07-encrypted-frames

Regarding the Meshlium configuration (gateway of the network), please refer to the Meshlium Technical Guide. In the “Application layer key management” section, you can see how to set up the nodes’ keys.



Node ID: node3

AES Secret Key: (16,24 or 32 characters) libeliumlibeliumlibelium

Figure: Encryption key setup

4.3. Device to Cloud encryption

This encryption mode is similar to the “Device to Gateway encryption” layer. However, instead of keeping the keys located in the gateway, they are stored in the Cloud. This allows to use different gateways (trusted or not trusted) as the information goes encrypted through them.

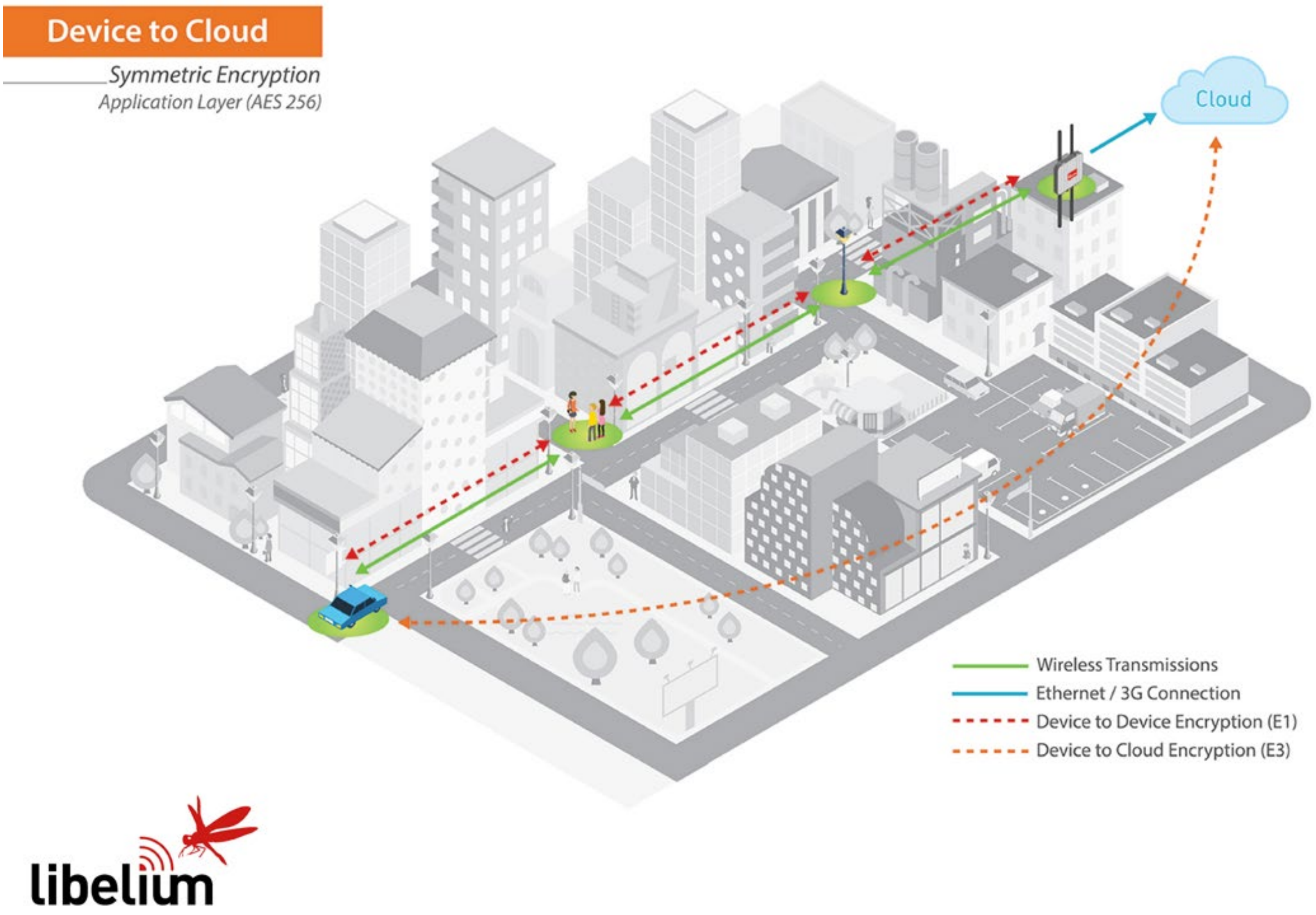


Figure: Device to cloud encryption.

The `encryptionToCloud()` function allows the user to enable/disable the “Device to Cloud” encryption feature. This feature must be enabled before calling the `encryptFrame()` function. Basically, a different “frame type” is used in order to differentiate the “Device to Cloud” encryption mode from the “Device to Gateway” encryption mode. The frame structure is equal for the two encryption modes.

Therefore, the “Device to Cloud” encryption process follows these steps:

- Step 1: Enable “Device to Cloud” encryption mode
- Step 2: Create a new Frame (ASCII or BINARY)
- Step 3: Encrypt the frame and use it as Encrypted Payload of the new Encrypted Frame

If the “Device to Cloud” encrypted frame is sent to Meshlium, it is stored inside the Meshlium database as it was received. It remains unencrypted. Then it should be sent to the Cloud server where the decryption process should be performed.

5. Code examples

In the Waspmote Development section you can find complete examples:
http://www.libelium.com/development/waspmote/examples

6. API changelog

Keep track of the software changes on this link:
www.libelium.com/development/waspmote/documentation/changelog/#Frame

7. Documentation changelog

From v7.5 to v7.6

- Added sensor fields for the Particle Matter sensor, Datasol Met and generic CAN Bus sensors

From v7.4 to v7.5

- Added sensor fields for Smart Water Xtreme
- Added additional sensor fields for Smart Agriculture Xtreme
- Changed Modbus variables types in the sensor fields table: from ‘uint16_t’ to ‘int’ type

From v7.3 to v7.4

- Added sensor fields for Smart Agriculture Xtreme

From v7.2 to v7.3

- Added frame fragmentation functions
- Added sensor fields for Smart Agriculture Xtreme

From v7.1 to v7.2

- Sensor fields table updated
- Binary frame format description improved

From v7.0 to v7.1

- Added new “Tiny” frame format for short-payload protocols
- Added new section for encrypted frames
- Added updates and fixes for new product generation modules
- Frame buffer size incremented from 150 to 255 bytes