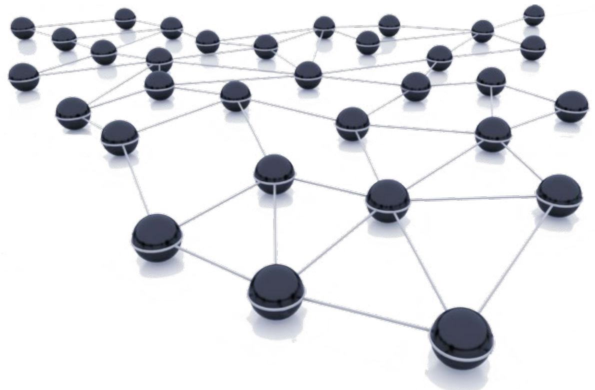
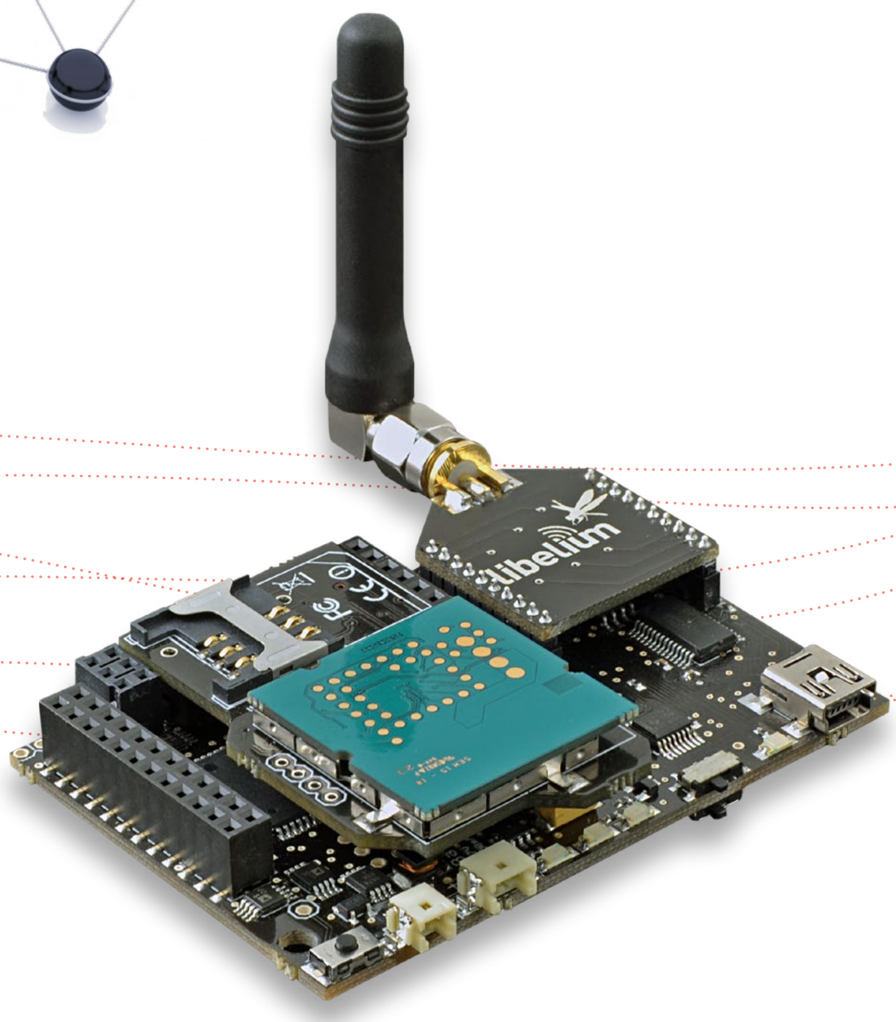
Waspmote Data Frame

Programming Guide





Document Version: v4.1 - 04/2013

INDEX

1. General Considerations

1.1. Waspmote Frame Files

WaspFrame.h, WaspFrame.cpp, WaspFrameConstants.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. API functions

Through this guide there are many examples of the WaspFrame class usage. In these examples, API 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 150) 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.
EXAMPLE_FRAME: defines an example frame type.
TIMEOUT_FRAME: defines a timeout frame type.
EVENT_FRAME: defines an event frame type.
ALARM_FRAME: defines an alarm frame type.
SERVICE1_FRAME: defines a service1 frame type.
SERVICE2_FRAME: defines a service2 frame type.

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

There are two kind of frames: ASCII and Binary.

2.1. ASCII Frame

These frames are supposed to facilitate the comprehension of the data to be sent. As the frame is composed by 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 1: 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	#	Temp: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 2: ASCII Frame example

A → Start Delimiter [3 Bytes]: It is composed by 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 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 Fields Byte [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 [10 Bytes]: This is at most a 10-Byte field which identifies each Waspmote device uniquely. The serial ID is get 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 [0Byte-16Bytes]: 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 16Bytes]. When the user do not want to give any identifier, the field remains empty between frame's separators: "##".

G → Frame sequence [1Byte-3Bytes]: 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 so as 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 by 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.

There are three types of ASCII sensor fields:

- **Simple Data:** Sensor field is composed by a unique data. The format is: "sensor_label: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 data composed by two or three values. The format is: "sensor_label: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 by 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 Fields	Serial ID	Waspmote ID	#	Sequence	Sensor_1	Sensor_2	...	Sensor_n

Figure 3: Binary Frame structure

2.2.1. Binary Header

The structure fields are described below with an example:

HEADER							PAYLOAD								
<=>	0x00	0x03	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 4: Binary Frame example

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

B → Frame Type Byte [1Byte]: 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 Fields Byte [1Byte]: This field specifies the number of sensor fields sent in the frame. This helps to calculate the frame length.

D → Separator [1Byte]: 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 [4Byte]: This is a 4-Byte field which identifies each Waspmote device uniquely. The serial ID is get 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 16Bytes]. When the user do not want to give any identifier, the field remains empty indicated by a unique '#' character.

G → Frame sequence [1Byte]: 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 by 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.

There are three types of Binary sensor fields:

- **Simple Data:** The sensor field is composed by a unique data. The format of this field is: the first byte codifies the sensor type. 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:

ID (1 Byte)	Byte1	Byte2	Byte3	Byte4
SENSOR_TCA	0x00	0x00	0xD8	0x41

Figure 5: Binary simple sensor field

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

- **Complex Data:** This is the format used to send data composed by 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 by both latitude and longitude floats, which means that 8 bytes are needed for both float values:

ID (1 Byte)	Byte1	Byte2	Byte3	Byte4	Byte1	Byte2	Byte3	Byte4
SENSOR_GPS	0x59	0x9D	0x26	0x42	0xE0	0x10	0x61	0xBF

Figure 6: Binary complex sensor field

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:

ID (1 Byte)	Length	Byte1 ('h')	Byte2 ('e')	Byte3 ('l')	Byte4 ('l')	Byte5 ('o')
SENSOR_STR	0x05	0x68	0x65	0x6C	0x6C	0x6F

Figure 7: Binary string sensor field

2.3. 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 Types				
Frame Type Byte		Decimal value	Identifier	Description
bit7	bit6-bit0			
0 (Binary)	0000000	0	Example	Regular frame for examples
	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
	…	6 to 99	…	Reserved types
	1100100	100	INITIAL_PACKET	Transmission packet to init a file Transmission
	1100101	101	ID_PACKET	Transmision packet to send the session ID to Waspmote
	1100110	102	DATA_PACKET	Transmision packet to send data to Meshlium
	1100111	103	ACK_PACKET	Transmision packet to sned ACK/NACK to Waspmote
	1101000	104	END_PACKET	Transmision packet to end the file transmision
	…	105 to 119	…	Reserved types
	1111000	120	delete_firmware	OTA packet to delete a firmware from boot.txt
	1111001	121	check_new_program	OTA packet to give starting information
	1111010	122	new_firmware_received	OTA packet to start receiving a new firmware
	1111011	123	new_firmware_packets	OTA packet to receive firmware packets
	1111100	124	new_firmware_end	OTA packet to end a firmware transmission
	1111101	125	upload_firmware	OTA packet to run a new firmware to Waspmote
	1111110	126	request_ID	OTA packet to request the mote ID
	1111111	127	request_bootlist	OTA packet to request the boot.txt list
1 (ASCII)	0000000	128	Example	Regular frame for examples
	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
	…	134 to 255	…	Reserved types

Figure 8: Frame types

2.4. 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 of 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	Units
				Binary	ASCII		Type of variable	Size per Field (Bytes)	Default Decimal Precision	
Gases	Carbon Monoxide	9229	SENSOR_CO	0	CO	1	float	4	3	voltage
	Carbon Dioxide	9230	SENSOR_CO2	1	CO2	1	float	4	3	voltage
	Oxygen	9231	SENSOR_O2	2	O2	1	float	4	3	voltage
	Methane	9232	SENSOR_CH4	3	CH4	1	float	4	3	voltage
	Liquefied Petroleum Gases	9234	SENSOR_LPG	4	LPG	1	float	4	3	voltage
	Ammonia	9233	SENSOR_NH3	5	NH3	1	float	4	3	voltage
	Air Pollutants 1	9235	SENSOR_AP1	6	AP1	1	float	4	3	voltage
	Air Pollutants 2	9236	SENSOR_AP2	7	AP2	1	float	4	3	voltage
	Solvent Vapors	9237	SENSOR_SV	8	SV	1	float	4	3	voltage
	Nitrogen Dioxide	9238	SENSOR_NO2	9	NO2	1	float	4	3	voltage
	Ozone	9258	SENSOR_O3	10	O3	1	float	4	3	voltage
	Hydrocarbons	9201	SENSOR_VOC	11	VOC	1	float	4	3	voltage
	Temperature Celsius	9203	SENSOR_TCA	12	TCA	1	float	4	2	° C
	Temperature Fahrenheit	9203	SENSOR_TFA	13	TFA	1	float	4	2	° F
	Humidity	9204	SENSOR_HUMA	14	HUMA	1	float	4	1	%RH
	Pressure atmospheric	9250	SENSOR_PA	15	PA	1	float	4	2	Kilo Pascales
Events	Pressure/Weight	9219	SENSOR_PW	16	PW	1	float	4	3	Ohms
	Bend	9218	SENSOR_BEND	17	BEND	1	float	4	3	Ohms
	Vibration	9221 / 9222	SENSOR_VBR	18	VBR	1	uint8_t	1	0	Open / Closed
	Hall Effect	9207	SENSOR_HALL	19	HALL	1	uint8_t	1	0	Open / Closed
	Liquid Presence	9243	SENSOR_LP	20	LP	1	uint8_t	1	0	Open / Closed
	Liquid Level	9239 / 9240 / 9242	SENSOR_LL	21	LL	1	uint8_t	1	0	Open / Closed
	Luminosity	9205	SENSOR_LUM	22	LUM	1	float	4	3	Ohms
	Presence	9212	SENSOR_PIR	23	PIR	1	uint8_t	1	0	presence / Not presence
	Stretch	9217	SENSOR_ST	24	ST	1	float	4	3	Ohms
Smart Cities	Microphone	9259	SENSOR_MCP	25	MCP	1	uint8_t	1	0	dBA
	Crack detection gauge	9321	SENSOR_CDG	26	CDG	1	uint8_t	1	0	true/false
	Crack propagation gauge	9322	SENSOR_CPG	27	CPG	1	float	4	3	Ohms
	Linear Displacement	9319	SENSOR_LD	28	LD	1	float	4	3	mm
	Dust	9320	SENSOR_DUST	29	DUST	1	float	4	3	mg/m^3
	Ultrasound	9246 / 9213	SENSOR_US	30	US	1	float	4	2	m
Parking	Magnetic Field	N/A	SENSOR_MF	31	MF	3	int	2	0	LSBs
	Parking Spot Status	N/A	SENSOR_PS	32	PS	1	uint8_t	1	0	"Occupied / Empty"
Agriculture	Temperature ºC (Sensirion)	9247	SENSOR_TCB	33	TCB	1	float	4	2	° C
	Temperature ºF (Sensirion)	9247	SENSOR_TFB	34	TFB	1	float	4	2	° F
	Humidity (Sensirion)	9247	SENSOR_HUMB	35	HUMB	1	float	4	1	%RH
	Soil Temperature	9255	SENSOR_SOILT	36	SOILT	1	float	4	2	ºC
	Soil Moisture	9248	SENSOR_SOIL	37	SOIL	1	float	4	2	Frequency
	Leaf Wetness	9249	SENSOR_LW	38	LW	1	uint8_t	1	0	%

	Sensor	Sensor Reference	Sensor TAG	SENSOR ID		Number Of Fields	Binary		ASCII	Units
				Binary	ASCII		Type of variable	Size per Field (Bytes)	Default Decimal Precision	
Agriculture	Solar Radiation	9251	SENSOR_PAR	39	PAR	1	float	4	2	µmol*m^{-2}*s^{-1}
	Ultraviolet Radiation	9257	SENSOR_UV	40	UV	1	float	4	2	µmol*m^{-2}*s^{-1}
	Trunk Diameter	9252	SENSOR_TD	41	TD	1	float	4	3	mm
	Stem Diameter	9253	SENSOR_SD	42	SD	1	float	4	3	mm
	Fruit Diameter	9254	SENSOR_FD	43	FD	1	float	4	3	mm
	Anemometer	9256	SENSOR_ANE	44	ANE	1	float	4	2	km/h
	Wind Vane	9256	SENSOR_WV	45	WV	1	uint8_t	1	N/A	Direction
	Pluviometer	9256	SENSOR_PLV	46	PLV	1	float	4	2	mm/min
Radiation	Geiger tube	N/A	SENSOR_RAD	47	RAD	1	float	4	6 or 0	uSv/h or cpm
Smart Metering	Current	9266	SENSOR_CU	48	CU	1	float	4	2	A
	Water flow	9296 / 9297 / 9298	SENSOR_WF	49	WF	1	float	4	3	l/min
	Load cell	9260 / 9261 / 9262	SENSOR_LC	50	LC	1	float	4	3	voltaje
	Distance Foil	9267 / 9268	SENSOR_DF	51	DF	1	float	4	3	Ohms
Aditional	Battery	N/A	SENSOR_BAT	52	BAT	1	uint8_t	1	0	%
	Global Positioning System	WGPS	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 or 4	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	° C
	Accelerometer	N/A	SENSOR_ACC	63	ACC	3	int	2	0	mg
	Millis	N/A	SENSOR_MILLIS	64	MILLIS	1	ulong	4	0	ms
Special	String	N/A	SENSOR_STR	65	STR	1	string	variable	N/A	N/A
Meshlium	Meshlium BT Scanner	N/A	SENSOR_MBT	66	MBT	1	string	variable	N/A	N/A
	Meshlium WiFi Scanner	N/A	SENSOR_MWIFI	67	MWIFI	1	string	variable	N/A	N/A
RFID	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

Figure 9: Field types

3. Usage

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

3.1. Setting the Waspmote Identifier

There is a function which allows the user to store the Waspmote ID in the EEPROM memory. This function is named `setID`. The Waspmote ID will be used to set the corresponding field in the frame's header when calling `createFrame` function.

Example of use:

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

3.2. Creating new Frames

The function in charge of creating a new frame is: `createFrame`. This function selects the frame mode:

- **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 get from the EEPROM memory that `setID` function has previously set:

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

- 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`, used to limit the maximum frame size, to MAX_FRAME constant. This constant defines a maximum **default size of 150 bytes** per frame. As this is the maximum possible value, it can be modified in WaspFrameConstants.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	
			No AES encryption	AES encryption
XBee – 802.15.4	Link Encrypted	@16bit Unicast	98 Bytes	93 Bytes
		@64bit Unicast	94 Bytes	77 Bytes
		Broadcast	95 Bytes	77 Bytes
	Link Unencrypted		100 Bytes	93 Bytes
XBee – 868			100 Bytes	93 Bytes
XBee – 900	Link Encrypted		80 Bytes	77 Bytes
	Link Unencrypted		100 Bytes	93 Bytes
XBee - Digimesh			73 Bytes	61 Bytes
XBee - ZigBee	Link Encrypted	@64bit Unicast	66 Bytes	61 Bytes
		Broadcast	84 Bytes	77 Bytes
	Link Unencrypted	@64bit Unicast	74 Bytes	61 Bytes
		Broadcast	92 Bytes	77 Bytes
Bluetooth – transparent connection			Limited by MAX_FRAME	Limited by MAX_FRAME
GPRS			Limited by MAX_FRAME	Limited by MAX_FRAME
3G			Limited by MAX_FRAME	Limited by MAX_FRAME
WiFi			Limited by MAX_FRAME	Limited by MAX_FRAME

Figure 10: Maximum frame size per protocol

Note: *MAX_FRAME is 100 bytes but can be changed by te user.*

The function prototypes are:

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

Set frame size depending on the protocol, addressing and encryption 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_900
XBEE_868

"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);

}
```

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 "EXAMPLE_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:

EXAMPLE_FRAME
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

3.5. Adding Sensor Fields

This is the function which appends new sensor fields to the frame. The first parameter is the sensor tag to identify the sensor to be added. 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 so many input possibilities.

Depending on the sensor field a specific type is needed for Binary frames. If a mismatch occurs, a message will appear through USB port. The sensor table shows the needed data type for each sensor.

Each call to this function appends a new field if there is enough space for the new field. If not, the field will not be attached.

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());

  // set frame fields (Temperature in Celsius sensor - float)
  frame.addSensor(SENSOR_IN_TEMP, (float) RTC.getTemperature());
}
```

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

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

Payload																															
S	T	R	:	S	T	R	I	N	G	#	B	A	T	:	8	7	#	I	N	_	T	E	M	P	:	2	7	.	2	5	#
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Figure 11: ASCII frame payload example

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

Payload									
SENSOR_STR	Length	"STRING"	SENSOR_BAT	0x57	SENSOR_IN_TEMP	0x00	0x00	0xDA	0x41
0	1	2-7	8	9	10	11	12	13	14

Figure 12: 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 WaspFrameConstants.h:

```
#define SENSOR_CO      0
#define SENSOR_CO2     1
#define SENSOR_O2      2
#define SENSOR_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 WaspFrameConstants.h:

```
prog_char   str_CO[]    PROGMEM = "CO";         // 0
prog_char   str_CO2[]   PROGMEM = "CO2";        // 1
prog_char   str_O2[]    PROGMEM = "O2";         // 2
prog_char   str_CH4[]   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:

- SENSOR_TABLE: This is a string table in order to define the sensor labels. For ASCII frames.
- 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.
- SENSOR_FIELD_TABLE: This is a uint8_t table which specifies the number of fields for each sensor.
- 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 actual Frame

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

Example of use:

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

4. Code examples

In the Waspmote Development section you can find complete examples:

http://www.libelium.com/development/waspmote/examples

5. Documentation changelog

- Added references to 3G/GPRS Board in section: Expansion Radio Board
- Added 3G/GPRS in table *maximum frame size per protocol*
- Added changes respect to maximum frame size for GPRS, 3G y BT in table *maximum frame size per protocol*
- Added changes respect to Serial ID in ASCII and Binary
- Added changes in tables binary Frame structure and binary Frame example