

```

1  /*
2  * Copyright (C) 2016 Libelium Comunicaciones Distribuidas S.L.
3  * http://www.libelium.com
4  *
5  * This program is free software: you can redistribute it and/or modify
6  * it under the terms of the GNU Lesser General Public License as published by
7  * the Free Software Foundation, either version 2.1 of the License, or
8  * (at your option) any later version.
9
10 * This program is distributed in the hope that it will be useful,
11 * but WITHOUT ANY WARRANTY; without even the implied warranty of
12 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
13 * GNU Lesser General Public License for more details.
14
15 * You should have received a copy of the GNU Lesser General Public License
16 * along with this program. If not, see <http://www.gnu.org/licenses/>.
17 *
18 * Version:          1.9
19 * Design:           David Gascón
20 * Implementation:   Yuri Carmona, Javier Siscart, Joaquín Ruiz
21 */
22 /* modificaciones crades per RUBÉN ALMANSA PER EXIT RESEARCH GROUP a partir de la
23    línia 2138 per els valors del Wasp mote
24    que corresponen als frames creats hi ha diversos tipus de frame*/
25 #ifndef __WPROGRAM_H__
26     #include <WaspClasses.h>
27 #endif
28
29 #include "WaspFrame.h"
30 #include "WaspFrameConstants.h"
31
32 // 2131 comencen les modificacions dels frames que volem enviar un per a cada waspmote
33
34
35 /// Constructors //////////////////////////////////////
36
37 WaspFrame::WaspFrame()
38 {
39     // store number zero to EEPROM in order to start
40     // with this first frame sequence number
41     storeSequence(0);
42
43     ///! This code should be used when Hibernate feature is used instead of the
44     ///! previous storeSequence(0) calling
45     ///~ if( !( digitalRead(RTC_INT_PIN_MON) &&
46     (Utils.readEEPROM(HIB_ADDR)==HIB_VALUE) ))
47     ///~ {
48     ///~     storeSequence(0);
49     ///~ }
50
51     // set default maximum frame size. It might be changed using 'setFrameSize'
52     // function when using an XBee module
53     _maxSize=MAX_FRAME;
54
55     // init waspmote ID attribute
56     frame.getID(_waspMoteID);
57 }
58
59
60 /// Public Methods //////////////////////////////////////
61
62
63 /*
64 * setFrameSize( size ) - set maximum frame size
65 *
66 * This function sets the class attribute to the parameter given to this
67 * function always it doesn't exceed the constant predefined in WaspFrame.h
68 * called MAX_FRAME
69 *
70 */
71 void WaspFrame::setFrameSize( uint8_t size )

```

```

72 {
73     if( size < MAX_FRAME)
74     {
75         // set new maximum size
76         _maxSize = size;
77     }
78     else
79     {
80         // input parameter exceeds the predefined constant
81         _maxSize = MAX_FRAME;
82     }
83
84
85
86
87 }
88
89
90 /*
91  * setFrameSize( protocol, linkEncryption, AESEncryption)
92  *
93  * This function MUST only be used when using an XBee module. This is the unique
94  * module which has size restrictions depending on the protocol, addressing, link
95  * encryption and AES encryption use.
96  *
97  * The possible values for protocol are:
98  * XBEE_802_15_4
99  * ZIGBEE
100  * DIGIMESH
101  * XBEE_900
102  * XBEE_868
103  *
104  * The possible values for linkEncryption are:
105  * ENABLED = 1
106  * DISABLED = 0
107  *
108  * The possible values for AESEncryption are:
109  * ENABLED = 1
110  * DISABLED = 0
111  *
112  */
113 void WaspFrame::setFrameSize( uint8_t protocol,
114                               uint8_t linkEncryption,
115                               uint8_t AESEncryption)
116 {
117     // call function prototype using a 64-bit addressing (the unique unicast
118     // possible for all XBee modules but the XBee-802.15.4)
119     return setFrameSize( protocol, UNICAST_64B, linkEncryption, AESEncryption);
120 }
121
122 /*
123  * setFrameSize( protocol, addressing, linkEncryption, AESEncryption)
124  *
125  * This function MUST only be used when using an XBee module. This is the unique
126  * module which has size restrictions depending on the protocol, addressing, link
127  * encryption and AES encryption use.
128  *
129  * The possible values for protocol are:
130  * XBEE_802_15_4
131  * ZIGBEE
132  * DIGIMESH
133  * XBEE_900
134  * XBEE_868
135  *
136  * The possible values for addressgin are:
137  * UNICAST_16B ---> for Unicast 16-bit addressing (only for XBee-802.15.4)
138  * UNICAST_64B ---> for Unicast 64-bit addressing
139  * BROADCAST_MODE ---> for Broadcast addressing
140  *
141  * The possible values for linkEncryption are:
142  * ENABLED = 1
143  * DISABLED = 0
144  *

```

```

145  * The possible values for AESEncryption are:
146  *   ENABLED = 1
147  *   DISABLED = 0
148  *
149  */
150  void WaspFrame::setFrameSize(   uint8_t protocol,
151                                uint8_t addressing,
152                                uint8_t linkEncryption,
153                                uint8_t AESEncryption)
154  {
155
156      /// AES disabled
157      if( AESEncryption == DISABLED )
158      {
159          switch (protocol)
160          {
161              /// XBEE_802_15_4 //////////////////////////////////
162              case XBEE_802_15_4:
163
164                  if( linkEncryption == DISABLED)
165                  {
166                      // XBEE_802 & Link Disabled & AES Disabled
167                      _maxSize = 100;
168                  }
169                  else if( linkEncryption == ENABLED)
170                  {
171                      if( addressing == UNICAST_16B )
172                      {
173                          // XBEE_802 & Unicast 16B & Link Enabled & AES Disabled
174                          _maxSize = 98;
175                      }
176                      else if( addressing == UNICAST_64B )
177                      {
178                          // XBEE_802 & Unicast 64B & Link Enabled & AES Disabled
179                          _maxSize = 94;
180                      }
181                      else if( addressing == BROADCAST_MODE )
182                      {
183                          // XBEE_802 & Broadcast & Link Enabled & AES
184                          Disabled
185                          _maxSize = 95;
186                      }
187                  }
188                  break;
189
190              /// ZIGBEE //////////////////////////////////
191              case ZIGBEE:
192
193                  if( linkEncryption == DISABLED)
194                  {
195                      if( addressing == UNICAST_64B )
196                      {
197                          // ZIGBEE & Unicast & Link Disabled & AES Disabled
198                          _maxSize = 74;
199                      }
200                      else if( addressing == BROADCAST_MODE )
201                      {
202                          // ZIGBEE & Broadcast & Link Disabled & AES
203                          Disabled
204                          _maxSize = 92;
205                      }
206                  }
207                  else if( linkEncryption == ENABLED)
208                  {
209                      if( addressing == UNICAST_64B )
210                      {
211                          // ZIGBEE & Unicast 64B & Link Enabled & AES Disabled
212                          _maxSize = 66;
213                      }
214                      else if( addressing == BROADCAST_MODE )
215                      {
216                          // ZIGBEE & Broadcast & Link Enabled & AES
217                          Disabled

```

[illegible]

```

287
288         if( linkEncryption == DISABLED)
289         {
290             if( addressing == UNICAST_64B )
291             {
292                 // ZIGBEE & Unicast & Link Disabled & AES Enabled
293                 _maxSize = ((74-10-strlen(_wasmoteID))/16)*16;
294             }
295             else if( addressing == BROADCAST_MODE )
296             {
297                 // ZIGBEE & Broadcast & Link Disabled & AES
298                 Enabled
299                 _maxSize = ((92-10-strlen(_wasmoteID))/16)*16;
300             }
301         }
302         else if( linkEncryption == ENABLED)
303         {
304             if( addressing == UNICAST_64B )
305             {
306                 // ZIGBEE & Unicast 64B & Link Enabled & AES Enabled
307                 _maxSize = ((66-10-strlen(_wasmoteID))/16)*16;
308             }
309             else if( addressing == BROADCAST_MODE )
310             {
311                 // ZIGBEE & Broadcast & Link Enabled & AES
312                 Enabled
313                 _maxSize = ((84-10-strlen(_wasmoteID))/16)*16;
314             }
315         }
316         break;
317
318     /// DIGIMESH //////////////////////////////////////
319     case DIGIMESH:
320
321         _maxSize = ((73-10-strlen(_wasmoteID))/16)*16;
322         break;
323
324     /// XBEE_900 //////////////////////////////////////
325     case XBEE_900:
326
327         if( linkEncryption == DISABLED)
328         {
329             _maxSize =
330             ((100-10-strlen(_wasmoteID))/16)*16;
331
332         }
333         else if( linkEncryption == ENABLED)
334         {
335             _maxSize =
336             ((80-10-strlen(_wasmoteID))/16)*16;
337         }
338
339         break;
340
341     /// XBEE_868 //////////////////////////////////////
342     case XBEE_868:
343
344         _maxSize =
345         ((100-10-strlen(_wasmoteID))/16)*16;
346         break;
347
348     default :
349         // No limit
350         _maxSize = MAX_FRAME;
351         break;
352 }
353
354 }
355
356 /// No limit
357 else
358 {
359     _maxSize = MAX_FRAME;
360 }

```

```

354
355 }
356
357
358
359
360 /*
361  * getFrameSize( ) - returns the maximum frame size previously set
362  *
363  *
364  */
365 uint8_t WaspFrame::getFrameSize( void )
366 {
367     return _maxSize;
368 }
369
370
371
372 /*
373  * createFrame ( ) - Initialize frame buffer
374  *
375  * Also, frame header bytes are initiliazed with default values
376  *
377  */
378 void WaspFrame::createFrame(uint8_t mode, char* moteID)
379 {
380     // set waspmote ID
381     frame.setID( moteID );
382
383     // create new frame
384     frame.createFrame( mode );
385 }
386
387
388
389 /*
390  * createFrame ( ) - Initialize frame buffer
391  *
392  * Also, frame header bytes are initiliazed with default values
393  *
394  */
395 void WaspFrame::createFrame(uint8_t mode)
396 {
397     // local variables
398     uint8_t sequence;
399     char str[16];
400
401     // store mode: ASCII or BINARY
402     _mode = mode;
403
404     // init buffer
405     memset( buffer, 0x00, sizeof(buffer) );
406
407     // init counter
408     numFields = 0;
409
410     // set frame delimiter
411     buffer[0] = '<';
412     buffer[1] = '=';
413     buffer[2] = '>';
414
415     uint8_t type;
416
417     // set type of frame depending on the frame mode
418     if( _mode == ASCII )
419     {
420         /** ASCII FRAME **/
421         type=B10000000;
422         buffer[3]= type;
423
424         // initialize 'number of fields' byte
425         buffer[4] = 0x00;
426

```

```

427 // set the '#' separator
428 buffer[5]='#';
429
430 // set serial ID
431 length = 6;
432
433 // _serial_id is read in main.cpp
434 snprintf(str, sizeof(str), "%lu", _serial_id);
435
436 for( uint16_t i=0 ; i<strlen(str) ; i++ )
437 {
438     // break if end of string
439     if( str[i] == '\0')
440     {
441         break;
442     }
443     else
444     {
445         buffer[length] = str[i];
446         length++;
447     }
448 }
449
450 // set separator '#'
451 buffer[length] = '#';
452 length++;
453
454 // set identifier
455 for( int i=0 ; i<16 ; i++ )
456 {
457     // break if end of string
458     if( _waspmoteID[i] == '\0')
459     {
460         break;
461     }
462     else
463     {
464         // if '#' character appears -> change it for '_'
465         if( _waspmoteID[i] == '#' )
466         {
467             buffer[length] = '_';
468         }
469         else
470         {
471             buffer[length] = _waspmoteID[i];
472         }
473         length++;
474     }
475 }
476
477 // set separator '#'
478 buffer[length] = '#';
479 length++;
480
481 // read sequence number from EEPROM
482 sequence = readSequence();
483
484 // convert from integer to string
485 char seqStr[4];
486 itoa(sequence, seqStr, 10);
487
488 // add sequence number
489 memcpy( &buffer[length], seqStr, strlen(seqStr));
490 length += strlen(seqStr);
491
492 // add separator '#'
493 buffer[length] = '#';
494 length++;
495
496 // increment and store the frame sequence number
497 sequence++;
498 storeSequence(sequence);
499

```

```

500         // now the frame is ready to be filled with new sensor values!
501     }
502     else
503     {
504         if (_mode == BINARY)
505         {
506             /** BINARY FRAME **/
507             type=B000000000;
508             buffer[3] = type;
509         }
510
511         // set serial ID
512         // _serial_id is read in main.cpp
513         char val[4];
514         memcpy(val, (const void*)&_serial_id, 4);
515
516         /*union {
517             unsigned long f;
518             char b[4];
519         } u;
520         u.b[3] = val[3];
521         u.b[2] = val[2];
522         u.b[1] = val[1];
523         u.b[0] = val[0];
524
525         USB.println(u.f);    */
526
527         // concatenate sensor name to frame string
528         buffer[5] = val[0];
529         buffer[6] = val[1];
530         buffer[7] = val[2];
531         buffer[8] = val[3];
532         length = 9;
533
534         // set identifier
535         for( int i=0 ; i<16 ; i++ )
536         {
537             // break if end of string
538             if( _waspmoteID[i] == '\0' )
539             {
540                 break;
541             }
542             else
543             {
544                 // if '#' character appears -> change it for '_'
545                 if( _waspmoteID[i] == '#' )
546                 {
547                     buffer[length] = '_';
548                 }
549                 else
550                 {
551                     buffer[length]=_waspmoteID[i];
552                 }
553                 length++;
554             }
555         }
556
557         // set separator '#'
558         buffer[length] = '#';
559         length++;
560
561         // read and set the sequence number to the frame
562         sequence=readSequence();
563         buffer[length]=sequence;
564         length++;
565
566         // initialize 'number of bytes' field
567         buffer[4] = length - 5;
568
569         sequence++;
570         storeSequence(sequence);
571     }
572

```



```

573 }
574
575
576
577
578
579 #ifndef WaspAES_h
580 /*
581  * createEncryptedFrame () - Create encrypted frame from previous created frame
582  * The inner 'frame.buffer' is used for encapsulating the new Waspnote frame.
583  * The structure of the encrypted frames is:
584  *
585  * |-----|
586  * | <=> | Frame Type | Num Bytes | ID secret | Wasp ID | # | Encrypted Frame |
587  * |-----|
588  *
589  * Where 'Encrypted Frame' is the original 'frame.buffer' which is encrypted
590  * using the specifications of this function: AES key size and password. ECB
591  * mode and ZEROS padding are always used. The resulting encrypted frame is
592  * stored in the same 'frame.buffer'
593  */
594 uint8_t WaspFrame::encryptFrame( uint16_t keySize, char* password )
595 {
596     // Variable for encrypted message's length
597     uint16_t encrypted_length;
598
599     // calculate encrypted length
600     encrypted_length = AES.sizeOfBlocks(frame.length);
601
602     if( encrypted_length > frame._maxSize )
603     {
604         return 0;
605     }
606
607     // Buffer for the encrypted message with enough memory space
608     uint8_t encrypted_message[encrypted_length];
609
610     // create Encrypted message
611     AES.encrypt( keySize
612                 , password
613                 , frame.buffer
614                 , frame.length
615                 , encrypted_message
616                 , ECB
617                 , ZEROS);
618
619     /// Create new frame with the correct structure
620     /**
621
622     |-----|
623     | <=> | Frame Type | Num Bytes | secretID | Wasp ID | # | Encrypted Frame |
624     |-----|
625     */
626
627     // define the frame type depending on the
628     // key size. ECB mode is always used for Meshlium.
629     uint8_t frame_type;
630
631     if( keySize == AES_128 )
632     {
633         frame_type = AES128_ECB_FRAME;
634     }
635     else if( keySize == AES_192 )
636     {
637         frame_type = AES192_ECB_FRAME;
638     }
639     else if( keySize == AES_256 )
640     {
641         frame_type = AES256_ECB_FRAME;
642     }
643     else
644     {
645         return 0;

```

```

646     }
647
648     // set serial ID
649     char val[4];
650     memcpy(val, (const void*)&_serial_id, 4);
651
652     // set frame delimiter
653     frame.buffer[0] = '<';
654     frame.buffer[1] = '=';
655     frame.buffer[2] = '>';
656     frame.buffer[3] = frame_type;
657     frame.buffer[4] = encrypted_length + 5 + strlen(frame._waspmoteID); // length
658     frame.buffer[5] = val[0]; // serial ID
659     frame.buffer[6] = val[1]; // serial ID
660     frame.buffer[7] = val[2]; // serial ID
661     frame.buffer[8] = val[3]; // serial ID
662
663     // temporal length of frame
664     uint16_t temp_length = 9;
665
666     // waspmote ID
667     for(uint16_t i = 0; i < strlen(frame._waspmoteID) ; i++)
668     {
669         frame.buffer[temp_length+i] = frame._waspmoteID[i];
670     }
671     temp_length += strlen(frame._waspmoteID);
672
673     // separator
674     frame.buffer[temp_length] = '#';
675     temp_length++;
676
677     // copy payload: encrypted message
678     for( uint16_t j = 0 ; j < encrypted_length; j++)
679     {
680         frame.buffer[temp_length+j] = encrypted_message[j];
681     }
682     temp_length += encrypted_length;
683
684     // set frame.length attribute
685     frame.length = temp_length;
686
687     // update attribute with the special frame type
688     _mode = ENCRYPTED_FRAME;
689
690     return 1;
691 }
692 #endif
693
694
695
696
697
698 /*
699  * setFrameType () - Set Frame Type
700  *
701  * The possible frame types to be selected are:
702  * EXAMPLE_FRAME
703  * TIMEOUT_FRAME
704  * EVENT_FRAME
705  * ALARM_FRAME
706  * SERVICE1_FRAME
707  * SERVICE2_FRAME
708  *
709  */
710 void WaspFrame::setFrameType(uint8_t type)
711 {
712     // set Frame Type in bits b5-b0 of correspondent field
713     buffer[3] |= type & B00111111;
714 }
715
716
717
718 /*

```

```

719     * showFrame (void) - Show current frame buffer
720     *
721     */
722 void WaspFrame::showFrame(void)
723 {
724     USB.secureBegin();
725
726     for(int i = 0; i < 31 ; i++)
727     {
728         printByte( '=', 0);
729     }
730     printByte( '\r', 0);
731     printByte( '\n', 0);
732
733
734     if( _mode == ASCII )
735     {
736         printString( "Current ASCII", 0);
737     }
738     else if( _mode == BINARY )
739     {
740         printString( "Current BINARY", 0);
741     }
742     else if( _mode == ENCRYPTED_FRAME )
743     {
744         printString( "Current ENCRYPTED", 0);
745     }
746     else return (void)0;
747
748     printString( " Frame:\r\n", 0);
749
750     printString( "Length: ", 0);
751     printIntegerInBase(length, 10, 0);
752     printByte( '\r', 0);
753     printByte( '\n', 0);
754
755     printString( "Frame Type: ", 0);
756     printIntegerInBase(buffer[3], 10, 0);
757     printByte( '\r', 0);
758     printByte( '\n', 0);
759
760     printString( "frame (HEX): ", 0);
761     for( uint16_t i= 0; i < length ; i++ )
762     {
763         puthex((char)buffer[i],0);
764     }
765
766     printByte( '\r', 0);
767     printByte( '\n', 0);
768
769     printString( "frame (STR): ", 0);
770     for( uint16_t i= 0; i < length ; i++ )
771     {
772         printByte( buffer[i], 0);
773     }
774
775     printByte( '\r', 0);
776     printByte( '\n', 0);
777
778     for(int i = 0; i < 31 ; i++)
779     {
780         printByte( '=', 0);
781     }
782
783     printByte( '\r', 0);
784     printByte( '\n', 0);
785
786     USB.secureEnd();
787 }
788
789
790
791 /*

```

```

792 * addSensor (type, value) - add sensor value to frame
793 *
794 * Parameters:
795 *   type : Refers to the type of sensor data
796 *   value : indicates the sensor value as a float
797 *
798 * Returns:
799 *   'length' of the composed frame when ok
800 *   -1 when the maximum length of the frame is reached
801 *
802 */
803 int8_t WaspFrame::addSensor(uint8_t type, int value)
804 {
805     char str[10];
806
807     if(_mode == ASCII)
808     {
809         // get name of sensor from table
810         char name[10];
811         strcpy_P(name, (char*)pgm_read_word(&(SENSOR_TABLE[type])));
812
813         // convert from integer to string
814         itoa( value, str, 10);
815
816         // check if new sensor value fits in the frame or not
817         // in the case the maximum length is reached, exit with error
818         // if not, then add the new sensor length to the total length
819         if(!checkLength( strlen(name) +
820                         strlen(str) +
821                         strlen(":") +
822                         strlen("#")    ))
823         {
824             return -1;
825         }
826
827         // create index for each element to be inserted in the sensor field
828         // 'index_1' is needed for adding the sensor tag
829         // 'index_2' is needed for adding ':'
830         // 'index_3' is needed for adding sensor value
831         // 'index_4' is needed for adding the separator '#'
832         int index_1 = length-strlen(name)-strlen(str)-strlen(":")-strlen("#");
833         int index_2 = length-strlen(str)-strlen(":")-strlen("#");
834         int index_3 = length-strlen(str)-strlen("#");
835         int index_4 = length-strlen("#");
836
837         // add sensor tag
838         memcpy ( &buffer[index_1], name, strlen(name) );
839
840         // add ':'
841         memcpy ( &buffer[index_2], ":", strlen(":") );
842
843         // add input string defined in 'str'
844         memcpy ( &buffer[index_3], str, strlen(str) );
845
846         // add separator '#'
847         memcpy ( &buffer[index_4], "#", strlen("#") );
848
849         // increment sensor fields counter
850         numFields++;
851
852         // set sensor fields counter
853         buffer[4] =
            numFields;
            defecte 4 //
854     }
855     else
856     {
857         // check if the data input type corresponds to the sensor
858         if (value<=255)
859         {
860             if( checkFields(type, TYPE_UINT8, 1) == -1 ) return -1;
861         }
862         else

```

```

863     {
864         if( checkFields(type, TYPE_INT, 1) == -1 ) return -1;
865     }
866
867     // set data bytes (in this case, int is two bytes)
868     char val[2];
869     memcpy(val,&value,2);
870
871     /*char val1 = value &0xFF;
872     char val2 = (value >> 8) &0xFF; */
873
874     //Check again (1 byte or 2 bytes)
875     uint8_t config;
876     config =(uint8_t)pgm_read_word(&(SENSOR_TYPE_TABLE[type]));
877
878     if (config == TYPE_INT)
879     {
880         // check if new sensor value fits
881         if(!checkLength(3))
882         {
883             return -1;
884         }
885
886         // concatenate sensor name to frame string
887         buffer[length-3] = (char)type;
888         buffer[length-2] = val[0];
889         buffer[length-1] = val[1];
890         buffer[length] = '\0';
891     }
892     else
893     {
894         // check if new sensor value fits
895         if(!checkLength(2))
896         {
897             return -1;
898         }
899
900         // concatenate sensor name to frame string
901         buffer[length-2] = (char)type;
902         buffer[length-1] = val[0];
903         buffer[length] = '\0';
904     }
905
906     // increment sensor fields counter
907     numFields++;
908     // update number of bytes field
909     buffer[4] = frame.length-5;
910
911 }
912
913 return length;
914 }
915
916
917
918 /*
919 * addSensor (type, value) - add sensor value to frame
920 *
921 * Parameters:
922 *   type : Refers to the type of sensor data
923 *   value : indicates the sensor value as an unsigned long
924 *
925 * Returns:
926 *   'length' of the composed frame when ok
927 *   -1 when the maximum length of the frame is reached
928 *
929 */
930 int8_t WaspFrame::addSensor(uint8_t type, unsigned long value)
931 {
932     char str[20];
933
934     if(_mode == ASCII)
935     {

```

```

936         // get name of sensor from table
937         char name[10];
938         strcpy_P(name, (char*)pgm_read_word(&(SENSOR_TABLE[type])));
939
940         // convert from integer to string
941         ultoa( value, str, 10);
942
943
944         // check if new sensor value fits in the frame or not
945         // in the case the maximum length is reached, exit with error
946         // if not, then add the new sensor length to the total length
947         if(!checkLength( strlen(name) +
948                         strlen(str) +
949                         strlen(":") +
950                         strlen("#") ))
951         {
952             return -1;
953         }
954
955         // create index for each element to be inserted in the sensor field
956         // 'index_1' is needed for adding the sensor tag
957         // 'index_2' is needed for adding ':'
958         // 'index_3' is needed for adding sensor value
959         // 'index_4' is needed for adding the separator '#'
960         int index_1 = length-strlen(name)-strlen(str)-strlen(":")-strlen("#");
961         int index_2 = length-strlen(str)-strlen(":")-strlen("#");
962         int index_3 = length-strlen(str)-strlen("#");
963         int index_4 = length-strlen("#");
964
965         // add sensor tag
966         memcpy ( &buffer[index_1], name, strlen(name) );
967
968         // add ':'
969         memcpy ( &buffer[index_2], ":", strlen(":") );
970
971         // add input string defined in 'str'
972         memcpy ( &buffer[index_3], str, strlen(str) );
973
974         // add separator '#'
975         memcpy ( &buffer[index_4], "#", strlen("#") );
976
977         // increment sensor fields counter
978         numFields++;
979
980         // set sensor fields counter
981         buffer[4] = numFields;
982     }
983     else
984     {
985         // check if the data input type corresponds to the sensor
986         if(checkFields(type, TYPE_ULONG, 1) == -1 ) return -1;
987
988         // set data bytes (in this case, double is 4...)
989         char val[4];
990         memcpy(val,&value,4);
991
992         /* Check correct copy
993         union {
994             unsigned long f;
995             char b[4];
996         } u;
997         u.b[3] = val[3];
998         u.b[2] = val[2];
999         u.b[1] = val[1];
1000         u.b[0] = val[0];
1001         delay(1);
1002         USB.println(u.f);*/
1003
1004         // check if new sensor value fits /1+4/
1005         if(!checkLength(5))
1006         {
1007             return -1;
1008         }

```

```

1009
1010         // concatenate sensor name to frame string
1011
1012         buffer[length-5] = (char)type;
1013         buffer[length-4] = val[0];
1014         buffer[length-3] = val[1];
1015         buffer[length-2] = val[2];
1016         buffer[length-1] = val[3];
1017         buffer[length] = '\\0';
1018
1019         // increment sensor fields counter
1020         numFields++;
1021         // update number of bytes field
1022         buffer[4] = frame.length-5;
1023
1024     }
1025
1026     return length;
1027 }
1028
1029
1030
1031
1032 /*
1033  * addSensor (type, value) - add sensor value to frame
1034  *
1035  * Parameters:
1036  *   type : Refers to the type of sensor data
1037  *   value : indicates the sensor value as a float
1038  *
1039  * Returns:
1040  *   'length' of the composed frame when ok
1041  *   -1 when the maximum length of the frame is reached
1042  *
1043  */
1044 int8_t WaspFrame::addSensor(uint8_t type, double value)
1045 {
1046     // get name of sensor from table
1047     char numDecimals;
1048     numDecimals =(uint8_t)pgm_read_word(&(DECIMAL_TABLE[type]));
1049
1050     return addSensor(type, value, numDecimals);
1051 }
1052
1053
1054
1055 /*
1056  * addSensor (type, value, N) - add sensor value to frame
1057  *
1058  * Parameters:
1059  *   type : Refers to the type of sensor data
1060  *   value : indicates the sensor value as a float
1061  *   N : number of decimals
1062  *
1063  * Returns:
1064  *   'length' of the composed frame when ok
1065  *   -1 when the maximum length of the frame is reached
1066  *
1067  */
1068 int8_t WaspFrame::addSensor(uint8_t type, double value, int N)
1069 {
1070     char str[20];
1071
1072     if(_mode == ASCII)
1073     {
1074         // convert from float to string
1075         dtostrf( value, N, N, str );
1076
1077         // get name of sensor from table
1078         char name[10];
1079         strcpy_P(name, (char*)pgm_read_word(&(SENSOR_TABLE[type])));
1080
1081         // check if new sensor value fits in the frame or not

```

```

1082 // in the case the maximum length is reached, exit with error
1083 // if not, then add the new sensor length to the total length
1084 if(!checkLength( strlen(name) +
1085                 strlen(str) +
1086                 strlen(":") +
1087                 strlen("#") ))
1088 {
1089     return -1;
1090 }
1091
1092 // create index for each element to be inserted in the sensor field
1093 // 'index_1' is needed for adding the sensor tag
1094 // 'index_2' is needed for adding ':'
1095 // 'index_3' is needed for adding sensor value
1096 // 'index_4' is needed for adding the separator '#'
1097 int index_1 = length-strlen(name)-strlen(str)-strlen(":")-strlen("#");
1098 int index_2 = length-strlen(str)-strlen(":")-strlen("#");
1099 int index_3 = length-strlen(str)-strlen("#");
1100 int index_4 = length-strlen("#");
1101
1102 // add sensor tag
1103 memcpy ( &buffer[index_1], name, strlen(name) );
1104
1105 // add ':'
1106 memcpy ( &buffer[index_2], ":", strlen(":") );
1107
1108 // add input string defined in 'str'
1109 memcpy ( &buffer[index_3], str, strlen(str) );
1110
1111 // add separator '#'
1112 memcpy ( &buffer[index_4], "#", strlen("#") );
1113
1114 // increment sensor fields counter
1115 numFields++;
1116
1117 // set sensor fields counter
1118 buffer[4]=numFields;
1119 }
1120 else
1121 {
1122     // check if the data input type corresponds to the sensor
1123     if(checkFields(type, TYPE_FLOAT, 1) == -1 ) return -1;
1124
1125     // set data bytes (in this case, double is 4...)
1126     char val[4];
1127     memcpy(val,&value,4);
1128
1129     /* Check correct copy
1130     union {
1131         float f;
1132         char b[4];
1133     } u;
1134     u.b[3] = val[3];
1135     u.b[2] = val[2];
1136     u.b[1] = val[1];
1137     u.b[0] = val[0];
1138     delay(1);
1139     USB.println(u.f);*/
1140
1141     // check if new sensor value fits /1+4/
1142     if(!checkLength(5))
1143     {
1144         return -1;
1145     }
1146
1147     // concatenate sensor name to frame string
1148
1149     buffer[length-5] = (char)type;
1150     buffer[length-4] = val[0];
1151     buffer[length-3] = val[1];
1152     buffer[length-2] = val[2];
1153     buffer[length-1] = val[3];
1154     buffer[length] = '\0';

```



```

1155
1156     // increment sensor fields counter
1157     numFields++;
1158     // update number of bytes field
1159     buffer[4] = frame.length-5;
1160
1161 }
1162
1163 return length;
1164 }
1165
1166
1167
1168 /*
1169  * addSensor (type, value) - add sensor value to frame
1170  *
1171  * Parameters:
1172  *   type : Refers to the type of sensor data
1173  *   value : indicates the sensor value as a float
1174  *
1175  * Returns:
1176  *   'length' of the composed frame when ok
1177  *   -1 when the maximum length of the frame is reached
1178  *
1179  */
1180 int8_t WaspFrame::addSensor(uint8_t type, char* str)
1181 {
1182     // calculate the buffer length to be created
1183     // depending on the input string to be copied to it
1184     uint16_t string_length = strlen(str)+1;
1185     const uint16_t MAX_SIZE = 400;
1186
1187     // limit the max string length to MAX_SIZE Bytes
1188     // to avoid running out of memory
1189     if ( string_length >= MAX_SIZE)
1190     {
1191         string_length = MAX_SIZE;
1192     }
1193
1194     // create aux buffer
1195     char aux_str[string_length];
1196
1197     // clear aux buffer
1198     memset( aux_str, 0x00, sizeof(aux_str) );
1199
1200     // copy string to aux buffer
1201     strncpy( aux_str, str, sizeof(aux_str)-1 );
1202
1203     // avoid '#' characters inside the string
1204     for( uint16_t i = 0; i < strlen(aux_str) ; i++)
1205     {
1206         if( aux_str[i] == '#' )
1207         {
1208             aux_str[i] = '_';
1209         }
1210     }
1211
1212     if(_mode == ASCII)
1213     {
1214         // get name of sensor from table
1215         char name[10];
1216         strcpy_P(name, (char*)pgm_read_word(&(SENSOR_TABLE[type])));
1217
1218         // check if new sensor value fits in the frame or not
1219         // in the case the maximum length is reached, exit with error
1220         // if not, then add the new sensor length to the total length
1221         if(!checkLength( strlen(name) +
1222                         strlen(aux_str) +
1223                         strlen(":") +
1224                         strlen("#") ))
1225         {
1226             return -1;
1227         }

```

```

1228
1229 // create index for each element to be inserted in the sensor field
1230 // 'index_1' is needed for adding the sensor tag
1231 // 'index_2' is needed for adding ':'
1232 // 'index_3' is needed for adding sensor value
1233 // 'index_4' is needed for adding the separator '#'
1234 int index_1 = length-strlen(name)-strlen(":")-strlen(aux_str)-strlen("#");
1235 int index_2 = length-strlen(":")-strlen(aux_str)-strlen("#");
1236 int index_3 = length-strlen(aux_str)-strlen("#");
1237 int index_4 = length-strlen("#");
1238
1239
1240 // add sensor tag
1241 memcpy ( &buffer[index_1], name, strlen(name) );
1242
1243 // add ':'
1244 memcpy ( &buffer[index_2], ":", strlen(":") );
1245
1246 // add input string defined in 'aux_str'
1247 memcpy ( &buffer[index_3], aux_str, strlen(aux_str) );
1248
1249 // add separator '#'
1250 memcpy ( &buffer[index_4], "#", strlen("#") );
1251
1252 // increment sensor fields counter
1253 numFields++;
1254
1255 // set sensor fields counter
1256 buffer[4] = numFields;
1257 }
1258 else
1259 {
1260 // check if the data input type corresponds to the sensor
1261 if(checkFields(type, TYPE_CHAR, 1) == -1 ) return -1;
1262
1263 // set data bytes (in this case, string, one byte per char) & length (1 byte)
1264 int8_t lng = strlen(aux_str);
1265
1266 // check if new sensor value fits (id + nbytes + str)
1267 if(!checkLength(2+strlen(aux_str)))
1268 {
1269     return -1;
1270 }
1271
1272 // concatenate sensor name to frame string
1273
1274 int len = length-2-strlen(aux_str);
1275 buffer[len] = (char)type;
1276 buffer[len+1] = lng;
1277 for (uint16_t j=len+2;j<length;j++)
1278 {
1279     buffer[j] = aux_str[j-2-len];
1280 }
1281 buffer[length] = '\\0';
1282
1283 // increment sensor fields counter
1284 numFields++;
1285 // update number of bytes field
1286 buffer[4] = frame.length-5;
1287
1288 }
1289
1290 return length;
1291 }
1292
1293
1294
1295
1296 /*
1297 * addSensor( type, val1, val2) - add sensor to frame
1298 *
1299 * Parameters:
1300 * type : Refers to the type of sensor data

```

```

1301  *   val1 : indicates the sensor value as a float
1302  *   val2 : indicates the sensor value as a float
1303  *
1304  * Returns:
1305  *   'length' of the composed frame when ok
1306  *   -1 when the maximum length of the frame is reached
1307  *
1308  */
1309 int8_t WaspFrame::addSensor(uint8_t type, double val1, double val2)
1310 {
1311
1312     char str1[20];
1313     char str2[20];
1314
1315     if(_mode == ASCII)
1316     {
1317         // get name of sensor from table
1318         char numDecimals;
1319         numDecimals =(uint8_t)pgm_read_word(&(DECIMAL_TABLE[type]));
1320
1321
1322         // convert from float to string
1323         dtostrf( val1, numDecimals, numDecimals, str1 );
1324         dtostrf( val2, numDecimals, numDecimals, str2 );
1325
1326         // get name of sensor from table
1327         char name[10];
1328         strcpy_P(name, (char*)pgm_read_word(&(SENSOR_TABLE[type])));
1329
1330         // check if new sensor value fits in the frame or not
1331         // in the case the maximum length is reached, exit with error
1332         // if not, then add the new sensor length to the total length
1333         if(!checkLength( strlen(name) +
1334                         strlen(str1) + strlen(str2) +
1335                         strlen(";") +
1336                         strlen(":") +
1337                         strlen("#")    ))
1338         {
1339             return -1;
1340         }
1341
1342         // create index for each element to be inserted in the sensor field
1343         // 'index_1' is needed for adding the sensor tag
1344         // 'index_2' is needed for adding ':'
1345         // 'index_3' is needed for adding sensor value in str1
1346         // 'index_4' is needed for adding ';'
1347         // 'index_5' is needed for adding sensor value in str2
1348         // 'index_6' is needed for adding the separator '#'
1349         int index_1 =
1350             length-strlen(name)-strlen(":")-strlen(str1)-strlen(";")-strlen(str2)-strlen("#");
1351         int index_2 =
1352             length-strlen(":")-strlen(str1)-strlen(";")-strlen(str2)-strlen("#");
1353         int index_3 = length-strlen(str1)-strlen(";")-strlen(str2)-strlen("#");
1354         int index_4 = length-strlen(";")-strlen(str2)-strlen("#");
1355         int index_5 = length-strlen(str2)-strlen("#");
1356         int index_6 = length-strlen("#");
1357
1358         // add sensor tag
1359         memcpy ( &buffer[index_1], name, strlen(name) );
1360
1361         // add ':'
1362         memcpy ( &buffer[index_2], ":", strlen(":") );
1363
1364         // add input string defined in 'str1'
1365         memcpy ( &buffer[index_3], str1, strlen(str1) );
1366
1367         // add ';'
1368         memcpy ( &buffer[index_4], ";", strlen(";") );
1369
1370         // add input string defined in 'str2'
1371         memcpy ( &buffer[index_5], str2, strlen(str2) );

```

```

1371
1372     // add separator '#'
1373     memcpy ( &buffer[index_6], "#", strlen("#") );
1374
1375     // increment sensor fields counter
1376     numFields++;
1377
1378     // set sensor fields counter
1379     buffer[4] = numFields;
1380 }
1381 else
1382 {
1383     // check if the data input type corresponds to the sensor
1384     if(checkFields(type, TYPE_FLOAT, 2) == -1 ) return -1;
1385
1386     // set data bytes (in this case, double is 4...)
1387     char valB1[4]; char valB2[4];
1388     memcpy(valB1,&val1,4);
1389     memcpy(valB2,&val2,4);
1390
1391     /*
1392     union{
1393         float f;
1394         char b[4];
1395     } u;
1396     u.b[3] = valB1[3];
1397     u.b[2] = valB1[2];
1398     u.b[1] = valB1[1];
1399     u.b[0] = valB1[0];
1400
1401     USB.println(u.f);
1402     */
1403
1404     // check if new sensor value fits /1+4+4/
1405     if(!checkLength(9))
1406     {
1407         return -1;
1408     }
1409
1410     // concatenate sensor name to frame string
1411
1412     buffer[length-9] = (char)type;
1413     buffer[length-8] = valB1[0];
1414     buffer[length-7] = valB1[1];
1415     buffer[length-6] = valB1[2];
1416     buffer[length-5] = valB1[3];
1417     buffer[length-4] = valB2[0];
1418     buffer[length-3] = valB2[1];
1419     buffer[length-2] = valB2[2];
1420     buffer[length-1] = valB2[3];
1421     buffer[length] = '\\0';
1422
1423     // increment sensor fields counter
1424     numFields++;
1425     // update number of bytes field
1426     buffer[4] = frame.length-5;
1427
1428 }
1429
1430 return length;
1431 }
1432
1433
1434 /*
1435 * addSensor( type, val1, val2) - add sensor to frame
1436 *
1437 * Parameters:
1438 *   type : Refers to the type of sensor data
1439 *   val1 : indicates the sensor value as a float
1440 *   val2 : indicates the sensor value as a float
1441 *
1442 * Returns:
1443 *   'length' of the composed frame when ok

```

```

1444     * -1 when the maximum length of the frame is reached
1445     *
1446     */
1447     int8_t WaspFrame::addSensor(uint8_t type, unsigned long val1, unsigned long val2)
1448     {
1449
1450         char str1[20];
1451         char str2[20];
1452
1453         if(_mode == ASCII)
1454         {
1455             // get name of sensor from table
1456             char name[10];
1457             strcpy_P(name, (char*)pgm_read_word(&(SENSOR_TABLE[type])));
1458
1459             // convert from integer to string
1460             ultoa( val1, str1, 10);
1461             ultoa( val2, str2, 10);
1462
1463
1464             // check if new sensor value fits in the frame or not
1465             // in the case the maximum length is reached, exit with error
1466             // if not, then add the new sensor length to the total length
1467             if(!checkLength( strlen(name) +
1468                             strlen(str1) + strlen(str2) +
1469                             strlen(";") +
1470                             strlen(":") +
1471                             strlen("#")    ))
1472             {
1473                 return -1;
1474             }
1475
1476             // create index for each element to be inserted in the sensor field
1477             // 'index_1' is needed for adding the sensor tag
1478             // 'index_2' is needed for adding ':'
1479             // 'index_3' is needed for adding sensor value in str1
1480             // 'index_4' is needed for adding ';'
1481             // 'index_5' is needed for adding sensor value in str2
1482             // 'index_6' is needed for adding the separator '#'
1483             int index_1 =
1484             length-strlen(name)-strlen(":")-strlen(str1)-strlen(";")-strlen(str2)-strlen("#");
1485             int index_2 =
1486             length-strlen(":")-strlen(str1)-strlen(";")-strlen(str2)-strlen("#");
1487             int index_3 = length-strlen(str1)-strlen(";")-strlen(str2)-strlen("#");
1488             int index_4 = length-strlen(";")-strlen(str2)-strlen("#");
1489             int index_5 = length-strlen(str2)-strlen("#");
1490             int index_6 = length-strlen("#");
1491
1492             // add sensor tag
1493             memcpy ( &buffer[index_1], name, strlen(name) );
1494
1495             // add ':'
1496             memcpy ( &buffer[index_2], ":", strlen(":") );
1497
1498             // add input string defined in 'str1'
1499             memcpy ( &buffer[index_3], str1, strlen(str1) );
1500
1501             // add ';'
1502             memcpy ( &buffer[index_4], ";", strlen(";") );
1503
1504             // add input string defined in 'str2'
1505             memcpy ( &buffer[index_5], str2, strlen(str2) );
1506
1507             // add separator '#'
1508             memcpy ( &buffer[index_6], "#", strlen("#") );
1509
1510             // increment sensor fields counter
1511             numFields++;
1512
1513             // set sensor fields counter
1514             buffer[4] = numFields;

```

```

1514     }
1515     else
1516     {
1517         // check if the data input type corresponds to the sensor
1518         if(checkFields(type, TYPE_ULONG, 2) == -1 ) return -1;
1519
1520         // set data bytes (in this case, double is 4...)
1521         char valB1[4]; char valB2[4];
1522         memcpy(valB1,&val1,4);
1523         memcpy(valB2,&val2,4);
1524
1525         // check if new sensor value fits /1+4+4/
1526         if(!checkLength(9))
1527         {
1528             return -1;
1529         }
1530
1531         // concatenate sensor name to frame string
1532
1533         buffer[length-9] = (char)type;
1534         buffer[length-8] = valB1[0];
1535         buffer[length-7] = valB1[1];
1536         buffer[length-6] = valB1[2];
1537         buffer[length-5] = valB1[3];
1538         buffer[length-4] = valB2[0];
1539         buffer[length-3] = valB2[1];
1540         buffer[length-2] = valB2[2];
1541         buffer[length-1] = valB2[3];
1542         buffer[length] = '\\0';
1543
1544         // increment sensor fields counter
1545         numFields++;
1546         // update number of bytes field
1547         buffer[4] = frame.length-5;
1548
1549     }
1550
1551     return length;
1552 }
1553
1554
1555
1556
1557 /*
1558  * addSensor( type, val1, val2, val3) - add sensor to frame
1559  *
1560  * Parameters:
1561  *   type : Refers to the type of sensor data
1562  *   val1 : indicates the sensor value as an uint8_t
1563  *   val2 : indicates the sensor value as an uint8_t
1564  *   val2 : indicates the sensor value as an uint8_t
1565  *
1566  * Returns:
1567  *   'length' of the composed frame when ok
1568  *   -1 when the maximum length of the frame is reached
1569  *
1570  * This function prototype is only thought for SENSOR_TIME and SENSOR_DATE fields
1571  *
1572  */
1573 int8_t WaspFrame::addSensor(uint8_t type, uint8_t val1, uint8_t val2, uint8_t val3)
1574 {
1575     char str1[10];
1576     char str2[10];
1577     char str3[10];
1578
1579     if(_mode == ASCII)
1580     {
1581         /// ASCII
1582
1583         // get name of sensor from table
1584         char name[10];
1585         strcpy_P(name, (char*)pgm_read_word(&(SENSOR_TABLE[type])));
1586

```

```

1587 // convert from integer to string
1588 itoa( val1, str1, 10);
1589 itoa( val2, str2, 10);
1590 itoa( val3, str3, 10);
1591
1592 // check if new sensor value fits in the frame or not
1593 // in the case the maximum length is reached, exit with error
1594 // if not, then add the new sensor length to the total length
1595 if(!checkLength( strlen(name) +
1596                 strlen(":") +
1597                 strlen(str1) +
1598                 strlen("-") +
1599                 strlen(str2) +
1600                 strlen("-") +
1601                 strlen(str3) +
1602                 strlen("#")    ))
1603 {
1604     return -1;
1605 }
1606
1607 // create index for each element to be inserted in the sensor field
1608 // 'index_1' is needed for adding the sensor tag
1609 // 'index_2' is needed for adding ':'
1610 // 'index_3' is needed for adding sensor value in str1
1611 // 'index_4' is needed for adding '-'
1612 // 'index_5' is needed for adding sensor value in str2
1613 // 'index_6' is needed for adding '-'
1614 // 'index_7' is needed for adding sensor value in str3
1615 // 'index_8' is needed for adding the separator '#'
1616 int index_1 =
1617 length-strlen(name)-strlen(":")-strlen(str1)-strlen("-")-strlen(str2)-strlen("-")-strlen(str3)-strlen("#");
1618 int index_2 =
1619 length-strlen(":")-strlen(str1)-strlen("-")-strlen(str2)-strlen("-")-strlen(str3)-strlen("#");
1620 int index_3 =
1621 length-strlen(str1)-strlen("-")-strlen(str2)-strlen("-")-strlen(str3)-strlen("#");
1622 int index_4 =
1623 length-strlen("-")-strlen(str2)-strlen("-")-strlen(str3)-strlen("#");
1624 int index_5 = length-strlen(str2)-strlen("-")-strlen(str3)-strlen("#");
1625 int index_6 = length-strlen("-")-strlen(str3)-strlen("#");
1626 int index_7 = length-strlen(str3)-strlen("#");
1627 int index_8 = length-strlen("#");
1628
1629 // add sensor tag
1630 memcpy ( &buffer[index_1], name, strlen(name) );
1631
1632 // add ':'
1633 memcpy ( &buffer[index_2], ":", strlen(":") );
1634
1635 // add input string defined in 'str1'
1636 memcpy ( &buffer[index_3], str1, strlen(str1) );
1637
1638 // add '-'
1639 memcpy ( &buffer[index_4], "-", strlen("-") );
1640
1641 // add input string defined in 'str2'
1642 memcpy ( &buffer[index_5], str2, strlen(str2) );
1643
1644 // add '-'
1645 memcpy ( &buffer[index_6], "-", strlen("-") );
1646
1647 // add input string defined in 'str3'
1648 memcpy ( &buffer[index_7], str3, strlen(str3) );
1649
1650 // add separator '#'
1651 memcpy ( &buffer[index_8], "#", strlen("#") );
1652
1653 // increment sensor fields counter
1654 numFields++;

```

```

1653         // set sensor fields counter
1654         buffer[4] = numFields;
1655     }
1656     else
1657     {
1658         /// BINARY
1659
1660         // check if the data input type corresponds to the sensor
1661         if(checkFields(type, TYPE_UINT8, 3) == -1 ) return -1;
1662
1663         // check if new sensor value fits 1+1+1+1
1664         if(!checkLength(4))
1665         {
1666             return -1;
1667         }
1668
1669         // concatenate sensor name to frame string
1670
1671         buffer[length-4] = (char)type;
1672         buffer[length-3] = val1;
1673         buffer[length-2] = val2;
1674         buffer[length-1] = val3;
1675         buffer[length] = '\0';
1676
1677         // increment sensor fields counter
1678         numFields++;
1679         // update number of bytes field
1680         buffer[4] = frame.length-5;
1681
1682     }
1683
1684     return length;
1685 }
1686
1687
1688
1689
1690
1691 /*
1692  * addSensor( type, val1, val2, val3, val4) - add sensor to frame
1693  *
1694  * Parameters:
1695  *   type : Refers to the type of sensor data
1696  *   val1 : indicates the sensor value as an uint8_t
1697  *   val2 : indicates the sensor value as an uint8_t
1698  *   val2 : indicates the sensor value as an uint8_t
1699  *   val2 : indicates the sensor value as an int
1700  *
1701  * Returns:
1702  *   'length' of the composed frame when ok
1703  *   -1 when the maximum length of the frame is reached
1704  *
1705  * This function prototype is only thought for SENSOR_TIME field
1706  *
1707  */
1708 int8_t WaspFrame::addSensor(uint8_t type, uint8_t val1, uint8_t val2, uint8_t val3,
1709 int val4)
1710 {
1711     char str1[10];
1712     char str2[10];
1713     char str3[10];
1714     char str4[10];
1715
1716     if(_mode == ASCII)
1717     {
1718         /// ASCII
1719
1720         // get name of sensor from table
1721         char name[10];
1722         strcpy_P(name, (char*)pgm_read_word(&(SENSOR_TABLE[type])));
1723
1724         // convert from integer to string
1725         itoa( val1, str1, 10);

```



```

1725     itoa( val2, str2, 10);
1726     itoa( val3, str3, 10);
1727     itoa( val4, str4, 10);
1728
1729     // check if new sensor value fits in the frame or not
1730     // in the case the maximum length is reached, exit with error
1731     // if not, then add the new sensor length to the total length
1732     if(!checkLength( strlen(name) +
1733                     strlen(":") +
1734                     strlen(str1) +
1735                     strlen("-") +
1736                     strlen(str2) +
1737                     strlen("-") +
1738                     strlen(str3) +
1739                     strlen("+") +
1740                     strlen(str4) +
1741                     strlen("#")    ))
1742     {
1743         return -1;
1744     }
1745
1746     // create index for each element to be inserted in the sensor field
1747     // 'index_1' is needed for adding the sensor tag
1748     // 'index_2' is needed for adding ':'
1749     // 'index_3' is needed for adding sensor value in str1
1750     // 'index_4' is needed for adding '-'
1751     // 'index_5' is needed for adding sensor value in str2
1752     // 'index_6' is needed for adding '-'
1753     // 'index_7' is needed for adding sensor value in str3
1754     // 'index_8' is needed for adding '+'
1755     // 'index_9' is needed for adding sensor value in str4
1756     // 'index_10' is needed for adding the separator '#'
1757     int index_1 =
length-strlen(name)-strlen(":")-strlen(str1)-strlen("-")-strlen(str2)-strlen("-")-strlen(str3)-strlen("+")-strlen(str4)-strlen("#");
1758     int index_2 =
length-strlen(":")-strlen(str1)-strlen("-")-strlen(str2)-strlen("-")-strlen(str3)-strlen("+")-strlen(str4)-strlen("#");
1759     int index_3 =
length-strlen(str1)-strlen("-")-strlen(str2)-strlen("-")-strlen(str3)-strlen("+")-strlen(str4)-strlen("#");
1760     int index_4 =
length-strlen("-")-strlen(str2)-strlen("-")-strlen(str3)-strlen("+")-strlen(str4)-strlen("#");
1761     int index_5 =
length-strlen(str2)-strlen("-")-strlen(str3)-strlen("+")-strlen(str4)-strlen("#");
1762     int index_6 =
length-strlen("-")-strlen(str3)-strlen("+")-strlen(str4)-strlen("#");
1763     int index_7 = length-strlen(str3)-strlen("+")-strlen(str4)-strlen("#");
1764     int index_8 = length-strlen("+")-strlen(str4)-strlen("#");
1765     int index_9 = length-strlen(str4)-strlen("#");
1766     int index_10 = length-strlen("#");
1767
1768
1769     // add sensor tag
1770     memcpy ( &buffer[index_1], name, strlen(name) );
1771
1772     // add ':'
1773     memcpy ( &buffer[index_2], ":", strlen(":") );
1774
1775     // add input string defined in 'str1'
1776     memcpy ( &buffer[index_3], str1, strlen(str1) );
1777
1778     // add '-'
1779     memcpy ( &buffer[index_4], "-", strlen("-") );
1780
1781     // add input string defined in 'str2'
1782     memcpy ( &buffer[index_5], str2, strlen(str2) );
1783
1784     // add '-'
1785     memcpy ( &buffer[index_6], "-", strlen("-") );
1786

```

```

1787 // add input string defined in 'str3'
1788 memcpy ( &buffer[index_7], str3, strlen(str3) );
1789
1790 if( val4>= 0 )
1791 {
1792     // add '+'
1793     memcpy ( &buffer[index_8], "+", strlen("+") );
1794 }
1795 else
1796 {
1797     // if negative then index_8 is useless because character '-' is
1798     // implicit within the number, so it is necessary to delete 1 Byte
1799     index_9 -= 1;
1800     index_10 -= 1;
1801 }
1802
1803 // add input string defined in 'str4'
1804 memcpy ( &buffer[index_9], str4, strlen(str4) );
1805
1806 // add separator '#'
1807 memcpy ( &buffer[index_10], "#", strlen("#") );
1808
1809 // increment sensor fields counter
1810 numFields++;
1811
1812 // set sensor fields counter
1813 buffer[4] = numFields;
1814 }
1815 else
1816 {
1817     /// BINARY
1818
1819     // As SENSOR_TIME is thought to be a 3-value field. This function permits to
1820     // add teh GMT value at the end of the time. But "checkFields" function must
1821     // expect 3 values because it is the original composition.
1822     if(checkFields(type, TYPE_UINT8, 3) == -1 ) return -1;
1823
1824     // check if new sensor value fits 1+1+1+1
1825     if(!checkLength(4))
1826     {
1827         return -1;
1828     }
1829
1830     // concatenate sensor name to frame string
1831
1832     buffer[length-4] = (char)type;
1833     buffer[length-3] = val1;
1834     buffer[length-2] = val2;
1835     buffer[length-1] = val3;
1836     buffer[length] = '\\0';
1837
1838     // increment sensor fields counter
1839     numFields++;
1840     // update number of bytes field
1841     buffer[4] = frame.length-5;
1842
1843 }
1844
1845 return length;
1846 }
1847
1848
1849
1850
1851
1852 /*
1853 * addSensor( type, val1, val2, val3) - add sensor to frame
1854 *
1855 * Parameters:
1856 * type : Refers to the type of sensor data
1857 * val1 : indicates the sensor value as an int (int16_t)
1858 * val2 : indicates the sensor value as an int (int16_t)
1859 * val2 : indicates the sensor value as an int (int16_t)

```

```

1860  *
1861  * Returns:
1862  *   'length' of the composed frame when ok
1863  *   -1 when the maximum length of the frame is reached
1864  *
1865  */
1866  int8_t WaspFrame::addSensor(uint8_t type, int val1,int val2,int val3)
1867  {
1868      char str1[10];
1869      char str2[10];
1870      char str3[10];
1871
1872      if(_mode == ASCII)
1873      {
1874          // get name of sensor from table
1875          char name[10];
1876          strcpy_P(name, (char*)pgm_read_word(&(SENSOR_TABLE[type])));
1877
1878          // convert from integer to string
1879          itoa( val1, str1, 10);
1880          itoa( val2, str2, 10);
1881          itoa( val3, str3, 10);
1882
1883          // check if new sensor value fits in the frame or not
1884          // in the case the maximum length is reached, exit with error
1885          // if not, then add the new sensor length to the total length
1886          if(!checkLength( strlen(name) +
1887                          strlen(":") +
1888                          strlen(str1) +
1889                          strlen(";") +
1890                          strlen(str2) +
1891                          strlen(";") +
1892                          strlen(str3) +
1893                          strlen("#")    ))
1894          {
1895              return -1;
1896          }
1897
1898          // create index for each element to be inserted in the sensor field
1899          // 'index_1' is needed for adding the sensor tag
1900          // 'index_2' is needed for adding ':'
1901          // 'index_3' is needed for adding sensor value in str1
1902          // 'index_4' is needed for adding ';'
1903          // 'index_5' is needed for adding sensor value in str2
1904          // 'index_6' is needed for adding ';'
1905          // 'index_7' is needed for adding sensor value in str3
1906          // 'index_8' is needed for adding the separator '#'
1907          int index_1 =
1908          length-strlen(name)-strlen(":")-strlen(str1)-strlen(";")-strlen(str2)-strlen("
;")-strlen(str3)-strlen("#");
1909          int index_2 =
1910          length-strlen(":")-strlen(str1)-strlen(";")-strlen(str2)-strlen(";")-strlen(st
r3)-strlen("#");
1911          int index_3 =
1912          length-strlen(str1)-strlen(";")-strlen(str2)-strlen(";")-strlen(str3)-strlen("
#");
1913          int index_4 =
1914          length-strlen(";")-strlen(str2)-strlen(";")-strlen(str3)-strlen("#");
1915          int index_5 = length-strlen(str2)-strlen(";")-strlen(str3)-strlen("#");
1916          int index_6 = length-strlen(";")-strlen(str3)-strlen("#");
1917          int index_7 = length-strlen(str3)-strlen("#");
1918          int index_8 = length-strlen("#");
1919
1920          // add sensor tag
1921          memcpy ( &buffer[index_1], name, strlen(name) );
1922
1923          // add ':'
1924          memcpy ( &buffer[index_2], ":", strlen(":") );
1925
1926          // add input string defined in 'str1'
1927          memcpy ( &buffer[index_3], str1, strlen(str1) );

```

```

1926         // add ';'
1927         memcpy ( &buffer[index_4], ";", strlen(";") );
1928
1929         // add input string defined in 'str2'
1930         memcpy ( &buffer[index_5], str2, strlen(str2) );
1931
1932         // add ';'
1933         memcpy ( &buffer[index_6], ";", strlen(";") );
1934
1935         // add input string defined in 'str3'
1936         memcpy ( &buffer[index_7], str3, strlen(str3) );
1937
1938         // add separator '#'
1939         memcpy ( &buffer[index_8], "#", strlen("#") );
1940
1941         // increment sensor fields counter
1942         numFields++;
1943
1944         // set sensor fields counter
1945         buffer[4]=numFields;
1946     }
1947     else
1948     {
1949         // check if the data input type corresponds to the sensor
1950         if( checkFields(type, TYPE_INT, 3) == -1 ) return -1;
1951
1952         // set data bytes (in this case, int is two bytes)
1953         char valC1[2];
1954         char valC2[2];
1955         char valC3[2];
1956         memcpy(valC1,&val1,2);
1957         memcpy(valC2,&val2,2);
1958         memcpy(valC3,&val3,2);
1959
1960         // check if new sensor value fits 1+2+2+2
1961         if(!checkLength(7))
1962         {
1963             return -1;
1964         }
1965
1966         // concatenate sensor name to frame string
1967
1968         buffer[length-7] = (char)type;
1969         buffer[length-6] = valC1[0];
1970         buffer[length-5] = valC1[1];
1971         buffer[length-4] = valC2[0];
1972         buffer[length-3] = valC2[1];
1973         buffer[length-2] = valC3[0];
1974         buffer[length-1] = valC3[1];
1975         buffer[length] = '\\0';
1976
1977         // increment sensor fields counter
1978         numFields++;
1979         // update number of bytes field
1980         buffer[4] = frame.length-5;
1981     }
1982 }
1983
1984 return length;
1985 }
1986
1987 /*
1988 *
1989 *
1990 *
1991 *
1992 *
1993 */
1994 int8_t WaspFrame::addSensor(uint8_t type, double val1,double val2,double val3)
1995 {
1996
1997     char str1[20];
1998     char str2[20];

```

```

1999     char str3[20];
2000
2001     if(_mode == ASCII)
2002     {
2003         // get name of sensor from table
2004         char numDecimals;
2005         numDecimals =(uint8_t)pgm_read_word(&(DECIMAL_TABLE[type]));
2006
2007         // convert from float to string
2008         dtostrf( val1,  numDecimals, numDecimals, str1 );
2009         dtostrf( val2,  numDecimals, numDecimals, str2 );
2010         dtostrf( val3,  numDecimals, numDecimals, str3 );
2011
2012         // get name of sensor from table
2013         char name[10];
2014         strcpy_P(name, (char*)pgm_read_word(&(SENSOR_TABLE[type])));
2015
2016         // check if new sensor value fits in the frame or not
2017         // in the case the maximum length is reached, exit with error
2018         // if not, then add the new sensor length to the total length
2019         if(!checkLength( strlen(name) +
2020                         strlen(str1) + strlen(str2) + strlen(str3) +
2021                         strlen(";") + strlen(";") + strlen(":") +
2022                         strlen("#")      ))
2023         {
2024             return -1;
2025         }
2026
2027         // create index for each element to be inserted in the sensor field
2028         // 'index_1' is needed for adding the sensor tag
2029         // 'index_2' is needed for adding ':'
2030         // 'index_3' is needed for adding sensor value in str1
2031         // 'index_4' is needed for adding ';'
2032         // 'index_5' is needed for adding sensor value in str2
2033         // 'index_6' is needed for adding ';'
2034         // 'index_7' is needed for adding sensor value in str3
2035         // 'index_8' is needed for adding the separator '#'
2036         int index_1 =
length-strlen(name)-strlen(":")-strlen(str1)-strlen(";")-strlen(str2)-strlen("
;")-strlen(str3)-strlen("#");
2037         int index_2 =
length-strlen(":")-strlen(str1)-strlen(";")-strlen(str2)-strlen(";")-strlen(st
r3)-strlen("#");
2038         int index_3 =
length-strlen(str1)-strlen(";")-strlen(str2)-strlen(";")-strlen(str3)-strlen("
#");
2039         int index_4 =
length-strlen(";")-strlen(str2)-strlen(";")-strlen(str3)-strlen("#");
2040         int index_5 = length-strlen(str2)-strlen(";")-strlen(str3)-strlen("#");
2041         int index_6 = length-strlen(";")-strlen(str3)-strlen("#");
2042         int index_7 = length-strlen(str3)-strlen("#");
2043         int index_8 = length-strlen("#");
2044
2045         // add sensor tag
2046         memcpy ( &buffer[index_1], name, strlen(name) );
2047
2048         // add ':'
2049         memcpy ( &buffer[index_2], ":", strlen(":") );
2050
2051         // add input string defined in 'str1'
2052         memcpy ( &buffer[index_3], str1, strlen(str1) );
2053
2054         // add ';'
2055         memcpy ( &buffer[index_4], ";", strlen(";") );
2056
2057         // add input string defined in 'str2'
2058         memcpy ( &buffer[index_5], str2, strlen(str2) );
2059
2060         // add ';'
2061         memcpy ( &buffer[index_6], ";", strlen(";") );
2062
2063         // add input string defined in 'str3'
2064         memcpy ( &buffer[index_7], str3, strlen(str3) );

```

```

2065
2066     // add separator '#'
2067     memcpy ( &buffer[index_8], "#", strlen("#") );
2068
2069     // increment sensor fields counter
2070     numFields++;
2071
2072     // set sensor fields counter
2073     buffer[4]=numFields;
2074 }
2075 else
2076 {
2077     // check if the data input type corresponds to the sensor
2078     if( checkFields(type, TYPE_FLOAT, 3) == -1 ) return -1;
2079
2080     // set data bytes (in this case, double is 4...)
2081     char valB1[4]; char valB2[4]; char valB3[4];
2082     memcpy(valB1,&val1,4);
2083     memcpy(valB2,&val2,4);
2084     memcpy(valB3,&val3,4);
2085     /*
2086     union {
2087         float f;
2088         char b[4];
2089     } u;
2090     u.b[3] = val[3];
2091     u.b[2] = val[2];
2092     u.b[1] = val[1];
2093     u.b[0] = val[0];
2094
2095     USB.println(u.f);
2096     */
2097     // check if new sensor value fits /1+4+4+4/
2098     if(!checkLength(13))
2099     {
2100         return -1;
2101     }
2102
2103     // concatenate sensor name to frame string
2104
2105     buffer[length-13] = (char)type;
2106     buffer[length-12] = valB1[0];
2107     buffer[length-11] = valB1[1];
2108     buffer[length-10] = valB1[2];
2109     buffer[length-9] = valB1[3];
2110     buffer[length-8] = valB2[0];
2111     buffer[length-7] = valB2[1];
2112     buffer[length-6] = valB2[2];
2113     buffer[length-5] = valB2[3];
2114     buffer[length-4] = valB3[0];
2115     buffer[length-3] = valB3[1];
2116     buffer[length-2] = valB3[2];
2117     buffer[length-1] = valB3[3];
2118     buffer[length] = '\\0';
2119
2120     // increment sensor fields counter
2121     numFields++;
2122     // update number of bytes field
2123     buffer[4] = frame.length-5;
2124
2125 }
2126
2127 return length;
2128 }
2129
2130
2131
2132 //afegit
2133 /*
2134 *
2135 *
2136 *
2137 *

```

```

2138  *
2139  */ // FRAME 1 creat Rubèn
2140  int8_t WaspFrame::addSensor(uint8_t type, double val1,double val2, double
val3,double val4,double val5,double val6,double val7,double val8) // huit valors
2141  {
2142
2143      char str1[20];
2144      char str2[20];
2145      char str3[20];
2146      char str4[20];
2147      char str5[20];
2148      char str6[20];
2149      char str7[20];
2150      char str8[20];
2151
2152      if(_mode == ASCII)
2153      {
2154          // get name of sensor from table
2155          char numDecimals;
2156          numDecimals =(uint8_t)pgm_read_word(&(DECIMAL_TABLE[type]));
2157
2158          // convert from float to string
2159          dtostrf( val1,  numDecimals, numDecimals, str1 );
2160          dtostrf( val2,  numDecimals, numDecimals, str2 );
2161          dtostrf( val3,  numDecimals, numDecimals, str3 );
2162          dtostrf( val4,  numDecimals, numDecimals, str4 );
2163          dtostrf( val5,  numDecimals, numDecimals, str5 );
2164          dtostrf( val6,  numDecimals, numDecimals, str6 );
2165          dtostrf( val7,  numDecimals, numDecimals, str7 );
2166          dtostrf( val8,  numDecimals, numDecimals, str8 );
2167
2168          // get name of sensor from table
2169          char name[10];
2170          strcpy_P(name, (char*)pgm_read_word(&(SENSOR_TABLE[type])));
2171
2172          // check if new sensor value fits in the frame or not
2173          // in the case the maximum length is reached, exit with error
2174          // if not, then add the new sensor length to the total length
2175          if(!checkLength( strlen(name) +
2176                          strlen(str1) + strlen(str2) + strlen(str3) +
2177                          strlen(str4) + strlen(str5) + strlen(str6) +
2178                          strlen(str7) + strlen(str8) +
2179                          strlen(";") + strlen(";") + strlen(";") +
2180                          strlen(";") + strlen(";") + strlen(";") +
2181                          strlen(";") + strlen(":") +
2182                          strlen("#")      ))
2183          {
2184              return -1;
2185          }
2186
2187          // create index for each element to be inserted in the sensor field
2188          // 'index_1' is needed for adding the sensor tag
2189          // 'index_2' is needed for adding ':'
2190          // 'index_3' is needed for adding sensor value in str1
2191          // 'index_4' is needed for adding ';'
2192          // 'index_5' is needed for adding sensor value in str2
2193          // 'index_6' is needed for adding ';'
2194          // 'index_7' is needed for adding sensor value in str3
2195          // 'index_8' is needed for adding the separator '#'
2196          /*int index_1 =
length-strlen(name)-strlen(":")-strlen(str1)-strlen(";")-strlen(str2)-strlen("
;")-strlen(str3)-strlen("#");
2197          int index_2 =
length-strlen(":")-strlen(str1)-strlen(";")-strlen(str2)-strlen(";")-strlen(st
r3)-strlen("#");
2198          int index_3 =
length-strlen(str1)-strlen(";")-strlen(str2)-strlen(";")-strlen(str3)-strlen("
#");
2199          int index_4 =
length-strlen(";")-strlen(str2)-strlen(";")-strlen(str3)-strlen("#");
2200          int index_5 = length-strlen(str2)-strlen(";")-strlen(str3)-strlen("#");
2201          int index_6 = length-strlen(";")-strlen(str3)-strlen("#");
2202          int index_7 = length-strlen(str3)-strlen("#");

```

```

2203     int index_8 = length-strlen("#");*/
2204
2205
2206     int index_1 =
length-strlen(name)-strlen(":")-strlen(str1)-strlen(";")-strlen(str2)-strlen("
;")-strlen(str3)-strlen(";")-strlen(str4)-strlen(";")-strlen(str5)-strlen(";")
-strlen(str6)-strlen(";")-strlen(str7)-strlen(";")-strlen(str8)-strlen("#");
2207     int index_2 =
length-strlen(":")-strlen(str1)-strlen(";")-strlen(str2)-strlen(";")-strlen(st
r3)-strlen(";")-strlen(str4)-strlen(";")-strlen(str5)-strlen(";")-strlen(str6)
-strlen(";")-strlen(str7)-strlen(";")-strlen(str8)-strlen("#");
2208     int index_3 =
length-strlen(str1)-strlen(";")-strlen(str2)-strlen(";")-strlen(str3)-strlen("
;")-strlen(str4)-strlen(";")-strlen(str5)-strlen(";")-strlen(str6)-strlen(";")
-strlen(str7)-strlen(";")-strlen(str8)-strlen("#");
2209     int index_4 =
length-strlen(";")-strlen(str2)-strlen(";")-strlen(str3)-strlen(";")-strlen(st
r4)-strlen(";")-strlen(str5)-strlen(";")-strlen(str6)-strlen(";")-strlen(str7)
-strlen(";")-strlen(str8)-strlen("#");
2210     int index_5 =
length-strlen(str2)-strlen(";")-strlen(str3)-strlen(";")-strlen(str4)-strlen("
;")-strlen(str5)-strlen(";")-strlen(str6)-strlen(";")-strlen(str7)-strlen(";")
-strlen(str8)-strlen("#");
2211     int index_6 =
length-strlen(";")-strlen(str3)-strlen(";")-strlen(str4)-strlen(";")-strlen(st
r5)-strlen(";")-strlen(str6)-strlen(";")-strlen(str7)-strlen(";")-strlen(str8)
-strlen("#");
2212     int index_7 =
length-strlen(str3)-strlen(";")-strlen(str4)-strlen(";")-strlen(str5)-strlen("
;")-strlen(str6)-strlen(";")-strlen(str7)-strlen(";")-strlen(str8)-strlen("#")
;
2213     int index_8 =
length-strlen(";")-strlen(str4)-strlen(";")-strlen(str5)-strlen(";")-strlen(st
r6)-strlen(";")-strlen(str7)-strlen(";")-strlen(str8)-strlen("#");
2214     int index_9 =
length-strlen(str4)-strlen(";")-strlen(str5)-strlen(";")-strlen(str6)-strlen("
;")-strlen(str7)-strlen(";")-strlen(str8)-strlen("#");
2215     int index_10 =
length-strlen(";")-strlen(str5)-strlen(";")-strlen(str6)-strlen(";")-strlen(st
r7)-strlen(";")-strlen(str8)-strlen("#");
2216     int index_11 =
length-strlen(str5)-strlen(";")-strlen(str6)-strlen(";")-strlen(str7)-strlen("
;")-strlen(str8)-strlen("#");
2217     int index_12 =
length-strlen(";")-strlen(str6)-strlen(";")-strlen(str7)-strlen(";")-strlen(st
r8)-strlen("#");
2218     int index_13 =
length-strlen(str6)-strlen(";")-strlen(str7)-strlen(";")-strlen(str8)-strlen("
#");
2219     int index_14 =
length-strlen(";")-strlen(str7)-strlen(";")-strlen(str8)-strlen("#");
2220     int index_15 =
length-strlen(str7)-strlen(";")-strlen(str8)-strlen("#");
2221     int index_16 = length-strlen(";")-strlen(str8)-strlen("#");
2222     int index_17 = length-strlen(str8)-strlen("#");
2223     int index_18 = length-strlen("#");
2224
2225
2226
2227     // add sensor tag
2228     memcpy ( &buffer[index_1], name, strlen(name) );
2229
2230     // add ':'
2231     memcpy ( &buffer[index_2], ":", strlen(":") );
2232
2233     // add input string defined in 'str1'
2234     memcpy ( &buffer[index_3], str1, strlen(str1) );
2235
2236     // add ';'
2237     memcpy ( &buffer[index_4], ";", strlen(";") );
2238
2239     // add input string defined in 'str2'
2240     memcpy ( &buffer[index_5], str2, strlen(str2) );

```



```

2241
2242 // add ';'
2243 memcpy ( &buffer[index_6], ";", strlen(";") );
2244
2245 // add input string defined in 'str3'
2246 memcpy ( &buffer[index_7], str3, strlen(str3) );
2247
2248 // add ';'
2249 memcpy ( &buffer[index_8], ";", strlen(";") );
2250
2251 // add input string defined in 'str3'
2252 memcpy ( &buffer[index_9], str4, strlen(str4) );
2253
2254 // add ';'
2255 memcpy ( &buffer[index_10], ";", strlen(";") );
2256
2257 // add input string defined in 'str3'
2258 memcpy ( &buffer[index_11], str5, strlen(str5) );
2259
2260 // add ';'
2261 memcpy ( &buffer[index_12], ";", strlen(";") );
2262
2263 // add input string defined in 'str3'
2264 memcpy ( &buffer[index_13], str6, strlen(str6) );
2265
2266 // add ';'
2267 memcpy ( &buffer[index_14], ";", strlen(";") );
2268
2269 // add input string defined in 'str3'
2270 memcpy ( &buffer[index_15], str7, strlen(str7) );
2271
2272 // add ';'
2273 memcpy ( &buffer[index_16], ";", strlen(";") );
2274
2275 // add input string defined in 'str3'
2276 memcpy ( &buffer[index_17], str8, strlen(str8) );
2277
2278 // add separator '#'
2279 memcpy ( &buffer[index_18], "#", strlen("#") );
2280
2281 // increment sensor fields counter
2282 numFields++;
2283
2284 // set sensor fields counter
2285 buffer[4]=numFields;
2286 }
2287 else
2288 {
2289 // check if the data input type corresponds to the sensor
2290 if( checkFields(type, TYPE_FLOAT, 8) == -1 ) return -1;
2291
2292 // set data bytes (in this case, double is 4...)
2293 char valB1[4]; char valB2[4]; char valB3[4]; char valB4[4]; char valB5[4]; char
valB6[4]; char valB7[4]; char valB8[4];
2294 memcpy(valB1,&val1,4);
2295 memcpy(valB2,&val2,4);
2296 memcpy(valB3,&val3,4);
2297 memcpy(valB4,&val4,4);
2298 memcpy(valB5,&val5,4);
2299 memcpy(valB6,&val6,4);
2300 memcpy(valB7,&val7,4);
2301 memcpy(valB8,&val8,4);
2302 /*
2303 union {
2304     float f;
2305     char b[4];
2306 } u;
2307 u.b[3] = val[3];
2308 u.b[2] = val[2];
2309 u.b[1] = val[1];
2310 u.b[0] = val[0];
2311
2312 USB.println(u.f);

```

```

2313     */
2314     // check if new sensor value fits /1+4+4+4/
2315     if(!checkLength(32))
2316     {
2317         return -1;
2318     }
2319
2320     // concatenate sensor name to frame string
2321
2322     buffer[length-32] = (char)type;
2323
2324
2325
2326
2327     buffer[length-31] = valB1[0];
2328     buffer[length-30] = valB1[1];
2329     buffer[length-29] = valB1[2];
2330     buffer[length-28] = valB1[3];
2331     buffer[length-27] = valB2[0];
2332     buffer[length-26] = valB2[1];
2333     buffer[length-25] = valB2[2];
2334     buffer[length-24] = valB2[3];
2335     buffer[length-23] = valB3[0];
2336     buffer[length-22] = valB3[1];
2337     buffer[length-21] = valB3[2];
2338     buffer[length-20] = valB3[3];
2339     buffer[length-19] = valB4[0];
2340     buffer[length-18] = valB4[1];
2341     buffer[length-17] = valB4[2];
2342     buffer[length-16] = valB4[3];
2343     buffer[length-15] = valB5[0];
2344     buffer[length-14] = valB5[1];
2345     buffer[length-13] = valB5[2];
2346     buffer[length-12] = valB5[3];
2347     buffer[length-12] = valB6[0];
2348     buffer[length-11] = valB6[1];
2349     buffer[length-10] = valB6[2];
2350     buffer[length-9] = valB6[3];
2351     buffer[length-8] = valB7[0];
2352     buffer[length-7] = valB7[1];
2353     buffer[length-6] = valB7[2];
2354     buffer[length-5] = valB7[3];
2355     buffer[length-4] = valB8[0];
2356     buffer[length-3] = valB8[1];
2357     buffer[length-2] = valB8[2];
2358     buffer[length-1] = valB8[3];
2359     buffer[length] = '\\0';
2360
2361     // increment sensor fields counter
2362     numFields++;
2363     // update number of bytes field
2364     buffer[4] = frame.length-5;
2365
2366 }
2367
2368 return length;
2369 }
2370 // creat per Rubén
2371 int8_t WaspFrame::addSensor(uint8_t type, double val1,double val2, int val3,int
val4,int val5,int val6,int val7,int val8) // vuit valors amb 6 int
2372 {
2373
2374     char str1[20];
2375     char str2[20];
2376     char str3[20];
2377     char str4[20];
2378     char str5[20];
2379     char str6[20];
2380     char str7[20];
2381     char str8[20];
2382
2383     if(_mode == ASCII)
2384     {

```

```

2385 // get name of sensor from table
2386 char numDecimals;
2387 numDecimals =(uint8_t)pgm_read_word(&(DECIMAL_TABLE[type]));
2388
2389 // convert from float to string
2390 dtostrf( val1, numDecimals, numDecimals, str1 );
2391 dtostrf( val2, numDecimals, numDecimals, str2 );
2392 itoa( val3, str3, 10);
2393 itoa( val4, str4, 10);
2394 itoa( val5, str5, 10);
2395 itoa( val6, str6, 10);
2396 itoa( val7, str7, 10);
2397 itoa( val8, str8, 10);
2398
2399 // get name of sensor from table
2400 char name[10];
2401 strcpy_P(name, (char*)pgm_read_word(&(SENSOR_TABLE[type])));
2402
2403 // check if new sensor value fits in the frame or not
2404 // in the case the maximum length is reached, exit with error
2405 // if not, then add the new sensor length to the total length
2406 if(!checkLength( strlen(name) +
2407                 strlen(str1) + strlen(str2) + strlen(str3) +
2408                 strlen(str4) + strlen(str5) + strlen(str6) +
2409                 strlen(str7) + strlen(str8) +
2410                 strlen(";") + strlen(";") + strlen(";") +
2411                 strlen(";") + strlen(";") + strlen(";") +
2412                 strlen(";") + strlen(":") +
2413                 strlen("#")    ))
2414 {
2415     return -1;
2416 }
2417
2418 // create index for each element to be inserted in the sensor field
2419 // 'index_1' is needed for adding the sensor tag
2420 // 'index_2' is needed for adding ':'
2421 // 'index_3' is needed for adding sensor value in str1
2422 // 'index_4' is needed for adding ';'
2423 // 'index_5' is needed for adding sensor value in str2
2424 // 'index_6' is needed for adding ';'
2425 // 'index_7' is needed for adding sensor value in str3
2426 // 'index_8' is needed for adding the separator '#'
2427 /*int index_1 =
length-strlen(name)-strlen(":")-strlen(str1)-strlen(";")-strlen(str2)-strlen("
;")-strlen(str3)-strlen("#");
int index_2 =
length-strlen(":")-strlen(str1)-strlen(";")-strlen(str2)-strlen(";")-strlen(st
r3)-strlen("#");
int index_3 =
length-strlen(str1)-strlen(";")-strlen(str2)-strlen(";")-strlen(str3)-strlen("
#");
int index_4 =
length-strlen(";")-strlen(str2)-strlen(";")-strlen(str3)-strlen("#");
int index_5 = length-strlen(str2)-strlen(";")-strlen(str3)-strlen("#");
int index_6 = length-strlen(";")-strlen(str3)-strlen("#");
int index_7 = length-strlen(str3)-strlen("#");
int index_8 = length-strlen("#");*/
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437 int index_1 =
length-strlen(name)-strlen(":")-strlen(str1)-strlen(";")-strlen(str2)-strlen("
;")-strlen(str3)-strlen(";")-strlen(str4)-strlen(";")-strlen(str5)-strlen(";")
-strlen(str6)-strlen(";")-strlen(str7)-strlen(";")-strlen(str8)-strlen("#");
2438 int index_2 =
length-strlen(":")-strlen(str1)-strlen(";")-strlen(str2)-strlen(";")-strlen(st
r3)-strlen(";")-strlen(str4)-strlen(";")-strlen(str5)-strlen(";")-strlen(str6)
-strlen(";")-strlen(str7)-strlen(";")-strlen(str8)-strlen("#");
2439 int index_3 =
length-strlen(str1)-strlen(";")-strlen(str2)-strlen(";")-strlen(str3)-strlen("
;")-strlen(str4)-strlen(";")-strlen(str5)-strlen(";")-strlen(str6)-strlen(";")
-strlen(str7)-strlen(";")-strlen(str8)-strlen("#");
2440 int index_4 =
length-strlen(";")-strlen(str2)-strlen(";")-strlen(str3)-strlen(";")-strlen(st

```

```

r4)-strlen(";")-strlen(str5)-strlen(";")-strlen(str6)-strlen(";")-strlen(str7)
-strlen(";")-strlen(str8)-strlen("#");
2441 int index_5 =
length-strlen(str2)-strlen(";")-strlen(str3)-strlen(";")-strlen(str4)-strlen("
;")-strlen(str5)-strlen(";")-strlen(str6)-strlen(";")-strlen(str7)-strlen(";")
-strlen(str8)-strlen("#");
2442 int index_6 =
length-strlen(";")-strlen(str3)-strlen(";")-strlen(str4)-strlen(";")-strlen(st
r5)-strlen(";")-strlen(str6)-strlen(";")-strlen(str7)-strlen(";")-strlen(str8)
-strlen("#");
2443 int index_7 =
length-strlen(str3)-strlen(";")-strlen(str4)-strlen(";")-strlen(str5)-strlen("
;")-strlen(str6)-strlen(";")-strlen(str7)-strlen(";")-strlen(str8)-strlen("#")
;
2444 int index_8 =
length-strlen(";")-strlen(str4)-strlen(";")-strlen(str5)-strlen(";")-strlen(st
r6)-strlen(";")-strlen(str7)-strlen(";")-strlen(str8)-strlen("#");
2445 int index_9 =
length-strlen(str4)-strlen(";")-strlen(str5)-strlen(";")-strlen(str6)-strlen("
;")-strlen(str7)-strlen(";")-strlen(str8)-strlen("#");
2446 int index_10 =
length-strlen(";")-strlen(str5)-strlen(";")-strlen(str6)-strlen(";")-strlen(st
r7)-strlen(";")-strlen(str8)-strlen("#");
2447 int index_11 =
length-strlen(str5)-strlen(";")-strlen(str6)-strlen(";")-strlen(str7)-strlen("
;")-strlen(str8)-strlen("#");
2448 int index_12 =
length-strlen(";")-strlen(str6)-strlen(";")-strlen(str7)-strlen(";")-strlen(st
r8)-strlen("#");
2449 int index_13 =
length-strlen(str6)-strlen(";")-strlen(str7)-strlen(";")-strlen(str8)-strlen("
#");
2450 int index_14 =
length-strlen(";")-strlen(str7)-strlen(";")-strlen(str8)-strlen("#");
2451 int index_15 =
length-strlen(str7)-strlen(";")-strlen(str8)-strlen("#");
2452 int index_16 = length-strlen(";")-strlen(str8)-strlen("#");
2453 int index_17 = length-strlen(str8)-strlen("#");
2454 int index_18 = length-strlen("#");
2455
2456
2457
2458 // add sensor tag
2459 memcpy ( &buffer[index_1], name, strlen(name) );
2460
2461 // add ':'
2462 memcpy ( &buffer[index_2], ":", strlen(":") );
2463
2464 // add input string defined in 'str1'
2465 memcpy ( &buffer[index_3], str1, strlen(str1) );
2466
2467 // add ';'
2468 memcpy ( &buffer[index_4], ";", strlen(";") );
2469
2470 // add input string defined in 'str2'
2471 memcpy ( &buffer[index_5], str2, strlen(str2) );
2472
2473 // add ';'
2474 memcpy ( &buffer[index_6], ";", strlen(";") );
2475
2476 // add input string defined in 'str3'
2477 memcpy ( &buffer[index_7], str3, strlen(str3) );
2478
2479 // add ';'
2480 memcpy ( &buffer[index_8], ";", strlen(";") );
2481
2482 // add input string defined in 'str3'
2483 memcpy ( &buffer[index_9], str4, strlen(str4) );
2484
2485 // add ';'
2486 memcpy ( &buffer[index_10], ";", strlen(";") );
2487
2488 // add input string defined in 'str3'

```

```

2489     memcpy ( &buffer[index_11], str5, strlen(str5) );
2490
2491     // add ';'
2492     memcpy ( &buffer[index_12], ";", strlen(";") );
2493
2494     // add input string defined in 'str3'
2495     memcpy ( &buffer[index_13], str6, strlen(str6) );
2496
2497     // add ';'
2498     memcpy ( &buffer[index_14], ";", strlen(";") );
2499
2500     // add input string defined in 'str3'
2501     memcpy ( &buffer[index_15], str7, strlen(str7) );
2502
2503     // add ';'
2504     memcpy ( &buffer[index_16], ";", strlen(";") );
2505
2506     // add input string defined in 'str3'
2507     memcpy ( &buffer[index_17], str8, strlen(str8) );
2508
2509     // add separator '#'
2510     memcpy ( &buffer[index_18], "#", strlen("#") );
2511
2512     // increment sensor fields counter
2513     numFields++;
2514
2515     // set sensor fields counter
2516     buffer[4]=numFields;
2517 }
2518 else
2519 {
2520     // check if the data input type corresponds to the sensor
2521     if( checkFields(type, TYPE_FLOAT, 8) == -1 ) return -1;
2522
2523     // set data bytes (in this case, double is 4...)
2524     char valB1[4]; char valB2[4]; char valB3[4];char valB4[4];char valB5[4];char
valB6[4];char valB7[4];char valB8[4];
2525     memcpy(valB1,&val1,4);
2526     memcpy(valB2,&val2,4);
2527     memcpy(valB3,&val3,4);
2528     memcpy(valB4,&val4,4);
2529     memcpy(valB5,&val5,4);
2530     memcpy(valB6,&val6,4);
2531     memcpy(valB7,&val7,4);
2532     memcpy(valB8,&val8,4);
2533     /*
2534     union {
2535         float f;
2536         char b[4];
2537     } u;
2538     u.b[3] = val[3];
2539     u.b[2] = val[2];
2540     u.b[1] = val[1];
2541     u.b[0] = val[0];
2542
2543     USB.println(u.f);
2544     */
2545     // check if new sensor value fits /1+4+4+4/
2546     if(!checkLength(32))
2547     {
2548         return -1;
2549     }
2550
2551     // concatenate sensor name to frame string
2552
2553     buffer[length-32] = (char)type;
2554
2555
2556
2557
2558     buffer[length-31] = valB1[0];
2559     buffer[length-30] = valB1[1];
2560     buffer[length-29] = valB1[2];

```

```

2561     buffer[length-28] = valB1[3];
2562     buffer[length-27] = valB2[0];
2563     buffer[length-26] = valB2[1];
2564     buffer[length-25] = valB2[2];
2565     buffer[length-24] = valB2[3];
2566     buffer[length-23] = valB3[0];
2567     buffer[length-22] = valB3[1];
2568     buffer[length-21] = valB3[2];
2569     buffer[length-20] = valB3[3];
2570     buffer[length-19] = valB4[0];
2571     buffer[length-18] = valB4[1];
2572     buffer[length-17] = valB4[2];
2573     buffer[length-16] = valB4[3];
2574     buffer[length-15] = valB5[0];
2575     buffer[length-14] = valB5[1];
2576     buffer[length-13] = valB5[2];
2577     buffer[length-12] = valB5[3];
2578     buffer[length-12] = valB6[0];
2579     buffer[length-11] = valB6[1];
2580     buffer[length-10] = valB6[2];
2581     buffer[length-9] = valB6[3];
2582     buffer[length-8] = valB7[0];
2583     buffer[length-7] = valB7[1];
2584     buffer[length-6] = valB7[2];
2585     buffer[length-5] = valB7[3];
2586     buffer[length-4] = valB8[0];
2587     buffer[length-3] = valB8[1];
2588     buffer[length-2] = valB8[2];
2589     buffer[length-1] = valB8[3];
2590     buffer[length] = '\\0';
2591
2592     // increment sensor fields counter
2593     numFields++;
2594     // update number of bytes field
2595     buffer[4] = frame.length-5;
2596
2597 }
2598
2599 return length;
2600 }
2601
2602
2603
2604
2605 /*// FRAME 2 creat Rubèn per la sensor board de Gasos enviem 6 valors "ANULAT"
2606
2607
2608
2609
2610 int8_t WaspFrame::addSensor(uint8_t type, double val1, double val2, double
val3,double val4,double val5,int val6)
2611 {
2612
2613     char str1[20];
2614     char str2[20];
2615     char str3[20];
2616     char str4[20];
2617     char str5[20];
2618     char str6[20];
2619
2620
2621     if(_mode == ASCII)
2622     {
2623         // get name of sensor from table
2624         char numDecimals;
2625         numDecimals =(uint8_t)pgm_read_word(&(DECIMAL_TABLE[type]));
2626
2627         // convert from float to string
2628         dtostrf( val1, numDecimals, numDecimals, str1 );
2629         dtostrf( val2, numDecimals, numDecimals, str2 );
2630         dtostrf( val3, numDecimals, numDecimals, str3 );
2631         dtostrf( val4, numDecimals, numDecimals, str4 );
2632         dtostrf( val5, numDecimals, numDecimals, str5 );

```

```

2633         itoa( val6, str6, 10);
2634
2635
2636         // get name of sensor from table
2637         char name[10];
2638         strcpy_P(name, (char*)pgm_read_word(&(SENSOR_TABLE[type])));
2639
2640         // check if new sensor value fits in the frame or not
2641         // in the case the maximum length is reached, exit with error
2642         // if not, then add the new sensor length to the total length
2643         if(!checkLength( strlen(name) +
2644                         strlen(str1) + strlen(str2) + strlen(str3) +
2645                         strlen(str4) + strlen(str5)+ strlen(str6)+
2646                         strlen(";") + strlen(";") + strlen(";") +
2647                         strlen(";") + strlen(";") +strlen(":")+
2648                         strlen("#")      ))
2649         {
2650             return -1;
2651         }
2652
2653         int index_1 =
length-strlen(name)-strlen(":")-strlen(str1)-strlen(";")-strlen(str2)-strlen("
;")-strlen(str3)-strlen(";")-strlen(str4)-strlen(";")-strlen(str5)-strlen(";")
-strlen(str6)-strlen("#");
2654         int index_2 =
length-strlen(":")-strlen(str1)-strlen(";")-strlen(str2)-strlen(";")-strlen(st
r3)-strlen(";")-strlen(str4)-strlen(";")-strlen(str5)-strlen(";")-strlen(str6)
-strlen("#");
2655         int index_3 =
length-strlen(str1)-strlen(";")-strlen(str2)-strlen(";")-strlen(str3)-strlen("
;")-strlen(str4)-strlen(";")-strlen(str5)-strlen(";")-strlen(str6)-strlen("#")
;
2656         int index_4 =
length-strlen(";")-strlen(str2)-strlen(";")-strlen(str3)-strlen(";")-strlen(st
r4)-strlen(";")-strlen(str5)-strlen(";")-strlen(str6)-strlen("#");
2657         int index_5 =
length-strlen(str2)-strlen(";")-strlen(str3)-strlen(";")-strlen(str4)-strlen("
;")-strlen(str5)-strlen(";")-strlen(str6)-strlen("#");
2658         int index_6 =
length-strlen(";")-strlen(str3)-strlen(";")-strlen(str4)-strlen(";")-strlen(st
r5)-strlen(";")-strlen(str6)-strlen("#");
2659         int index_7 =
length-strlen(str3)-strlen(";")-strlen(str4)-strlen(";")-strlen(str5)-strlen("
;")-strlen(str6)-strlen("#");
2660         int index_8 =
length-strlen(";")-strlen(str4)-strlen(";")-strlen(str5)-strlen(";")-strlen(st
r6)-strlen("#");
2661         int index_9 =
length-strlen(str4)-strlen(";")-strlen(str5)-strlen(";")-strlen(str6)-strlen("
#");
2662         int index_10 =
length-strlen(";")-strlen(str5)-strlen(";")-strlen(str6)-strlen("#");
2663         int index_11 = length-strlen(str5)-strlen(";")-strlen(str6)-strlen("#");
2664         int index_12 = length-strlen(";")-strlen(str6)-strlen("#");
2665         int index_13 = length-strlen(str6)-strlen("#");
2666         int index_14 = strlen("#");
2667
2668
2669
2670         // add sensor tag
2671         memcpy ( &buffer[index_1], name, strlen(name) );
2672
2673         // add ':'
2674         memcpy ( &buffer[index_2], ":", strlen(":") );
2675
2676         // add input string defined in 'str1'
2677         memcpy ( &buffer[index_3], str1, strlen(str1) );
2678
2679         // add ';'
2680         memcpy ( &buffer[index_4], ";", strlen(";") );
2681
2682         // add input string defined in 'str2'
2683         memcpy ( &buffer[index_5], str2, strlen(str2) );

```

```

2684
2685 // add ';'
2686 memcpy ( &buffer[index_6], ";", strlen(";") );
2687
2688 // add input string defined in 'str3'
2689 memcpy ( &buffer[index_7], str3, strlen(str3) );
2690
2691 // add ';'
2692 memcpy ( &buffer[index_8], ";", strlen(";") );
2693
2694 // add input string defined in 'str3'
2695 memcpy ( &buffer[index_9], str4, strlen(str4) );
2696
2697 // add ';'
2698 memcpy ( &buffer[index_10], ";", strlen(";") );
2699
2700 // add input string defined in 'str3'
2701 memcpy ( &buffer[index_11], str5, strlen(str5) );
2702 // add ';'
2703 memcpy ( &buffer[index_12], ";", strlen(";") );
2704
2705 // add input string defined in 'str3'
2706 memcpy ( &buffer[index_13], str6, strlen(str6) );
2707
2708 // add separator '#'
2709 memcpy ( &buffer[index_14], "#", strlen("#") );
2710
2711 // increment sensor fields counter
2712 numFields++;
2713
2714 // set sensor fields counter
2715 buffer[4]=numFields;
2716 }
2717 else
2718 {
2719 // check if the data input type corresponds to the sensor
2720 if( checkFields(type, TYPE_FLOAT, 3) == -1 ) return -1;
2721
2722 // set data bytes (in this case, double is 4...)
2723 char valB1[4]; char valB2[4]; char valB3[4];
2724 memcpy(valB1,&val1,4);
2725 memcpy(valB2,&val2,4);
2726 memcpy(valB3,&val3,4);
2727 /*
2728 union {
2729     float f;
2730     char b[4];
2731 } u;
2732 u.b[3] = val[3];
2733 u.b[2] = val[2];
2734 u.b[1] = val[1];
2735 u.b[0] = val[0];
2736
2737 USB.println(u.f);
2738 //
2739 // check if new sensor value fits /1+4+4+4/
2740 if(!checkLength(13))
2741 {
2742     return -1;
2743 }
2744
2745 // concatenate sensor name to frame string
2746
2747 buffer[length-13] = (char)type;
2748 buffer[length-12] = valB1[0];
2749 buffer[length-11] = valB1[1];
2750 buffer[length-10] = valB1[2];
2751 buffer[length-9] = valB1[3];
2752 buffer[length-8] = valB2[0];
2753 buffer[length-7] = valB2[1];
2754 buffer[length-6] = valB2[2];
2755 buffer[length-5] = valB2[3];
2756 buffer[length-4] = valB3[0];

```



```

2757     buffer[length-3] = valB3[1];
2758     buffer[length-2] = valB3[2];
2759     buffer[length-1] = valB3[3];
2760     buffer[length] = '\0';
2761
2762     // increment sensor fields counter
2763     numFields++;
2764     // update number of bytes field
2765     buffer[4] = frame.length-5;
2766
2767 }
2768
2769 return length;
2770 }*/
2771
2772 // Frame 3 creat Rubèn EVENTS  PORTA
2773
2774 int8_t WaspFrame::addSensor(uint8_t type, double val1, double val2, double val3, int
val4, int val5, int val6, int val7)
2775 {
2776
2777     char str1[20];
2778     char str2[20];
2779     char str3[20];
2780     char str4[20];
2781     char str5[20];
2782     char str6[20];
2783     char str7[20];
2784
2785     if(_mode == ASCII)
2786     {
2787         // get name of sensor from table
2788         char numDecimals;
2789         numDecimals =(uint8_t)pgm_read_word(&(DECIMAL_TABLE[type]));
2790
2791         // convert from float to string
2792         dtostrf( val1, numDecimals, numDecimals, str1 );
2793         dtostrf( val2, numDecimals, numDecimals, str2 );
2794         dtostrf( val3, numDecimals, numDecimals, str3 );
2795         itoa( val4, str4, 10);
2796         itoa( val5, str5, 10);
2797         itoa( val6, str6, 10);
2798         itoa( val7, str7, 10);
2799
2800         // get name of sensor from table
2801         char name[10];
2802         strcpy_P(name, (char*)pgm_read_word(&(SENSOR_TABLE[type])));
2803
2804         // check if new sensor value fits in the frame or not
2805         // in the case the maximum length is reached, exit with error
2806         // if not, then add the new sensor length to the total length
2807         if(!checkLength( strlen(name) +
2808                         strlen(str1) + strlen(str2) + strlen(str3) +
2809                         strlen(str4) + strlen(str5) + strlen(str6) +
2810                         strlen(str7) +
2811                         strlen(";") + strlen(";") + strlen(";") +
2812                         strlen(";") + strlen(";") + strlen(";") +
2813                         strlen(":") +
2814                         strlen("#")      ))
2815         {
2816             return -1;
2817         }
2818
2819
2820
2821         int index_1 =
length-strlen(name)-strlen(":")-strlen(str1)-strlen(";")-strlen(str2)-strlen("
;")-strlen(str3)-strlen(";")-strlen(str4)-strlen(";")-strlen(str5)-strlen(";")
-strlen(str6)-strlen(";")-strlen(str7)-strlen("#");
2822         int index_2 =
length-strlen(":")-strlen(str1)-strlen(";")-strlen(str2)-strlen(";")-strlen(st
r3)-strlen(";")-strlen(str4)-strlen(";")-strlen(str5)-strlen(";")-strlen(str6)
-strlen(";")-strlen(str7)-strlen("#");

```

```

2823     int index_3 =
length-strlen(str1)-strlen(";")-strlen(str2)-strlen(";")-strlen(str3)-strlen("
;")-strlen(str4)-strlen(";")-strlen(str5)-strlen(";")-strlen(str6)-strlen(";")
-strlen(str7)-strlen("#");
2824     int index_4 =
length-strlen(";")-strlen(str2)-strlen(";")-strlen(str3)-strlen(";")-strlen(st
r4)-strlen(";")-strlen(str5)-strlen(";")-strlen(str6)-strlen(";")-strlen(str7)
-strlen("#");
2825     int index_5 =
length-strlen(str2)-strlen(";")-strlen(str3)-strlen(";")-strlen(str4)-strlen("
;")-strlen(str5)-strlen(";")-strlen(str6)-strlen(";")-strlen(str7)-strlen("#")
;
2826     int index_6 =
length-strlen(";")-strlen(str3)-strlen(";")-strlen(str4)-strlen(";")-strlen(st
r5)-strlen(";")-strlen(str6)-strlen(";")-strlen(str7)-strlen("#");
2827     int index_7 =
length-strlen(str3)-strlen(";")-strlen(str4)-strlen(";")-strlen(str5)-strlen("
;")-strlen(str6)-strlen(";")-strlen(str7)-strlen("#");
2828     int index_8 =
length-strlen(";")-strlen(str4)-strlen(";")-strlen(str5)-strlen(";")-strlen(st
r6)-strlen(";")-strlen(str7)-strlen("#");
2829     int index_9 =
length-strlen(str4)-strlen(";")-strlen(str5)-strlen(";")-strlen(str6)-strlen("
;")-strlen(str7)-strlen("#");
2830     int index_10 =
length-strlen(";")-strlen(str5)-strlen(";")-strlen(str6)-strlen(";")-strlen(st
r7)-strlen("#");
2831     int index_11 =
length-strlen(str5)-strlen(";")-strlen(str6)-strlen(";")-strlen(str7)-strlen("
#");
2832     int index_12 =
length-strlen(";")-strlen(str6)-strlen(";")-strlen(str7)-strlen("#");
2833     int index_13 =
length-strlen(str6)-strlen(";")-strlen(str7)-strlen("#");
2834     int index_14 = length-strlen(";")-strlen(str7)-strlen("#");
2835     int index_15 = length-strlen(str7)-strlen("#");
2836     int index_16 = length-strlen("#");
2837
2838
2839
2840     // add sensor tag
2841     memcpy ( &buffer[index_1], name, strlen(name) );
2842
2843     // add ':'
2844     memcpy ( &buffer[index_2], ":", strlen(":") );
2845
2846     // add input string defined in 'str1'
2847     memcpy ( &buffer[index_3], str1, strlen(str1) );
2848
2849     // add ';'
2850     memcpy ( &buffer[index_4], ";", strlen(";") );
2851
2852     // add input string defined in 'str2'
2853     memcpy ( &buffer[index_5], str2, strlen(str2) );
2854
2855     // add ';'
2856     memcpy ( &buffer[index_6], ";", strlen(";") );
2857
2858     // add input string defined in 'str3'
2859     memcpy ( &buffer[index_7], str3, strlen(str3) );
2860
2861     // add ';'
2862     memcpy ( &buffer[index_8], ";", strlen(";") );
2863
2864     // add input string defined in 'str3'
2865     memcpy ( &buffer[index_9], str4, strlen(str4) );
2866
2867     // add ';'
2868     memcpy ( &buffer[index_10], ";", strlen(";") );
2869
2870     // add input string defined in 'str3'
2871     memcpy ( &buffer[index_11], str5, strlen(str5) );
2872

```

```

2873 // add ';'
2874 memcpy ( &buffer[index_12], ";", strlen(";") );
2875
2876 // add input string defined in 'str3'
2877 memcpy ( &buffer[index_13], str6, strlen(str6) );
2878
2879 // add ';'
2880 memcpy ( &buffer[index_14], ";", strlen(";") );
2881
2882 // add input string defined in 'str3'
2883 memcpy ( &buffer[index_15], str7, strlen(str7) );
2884
2885
2886
2887 // add separator '#'
2888 memcpy ( &buffer[index_16], "#", strlen("#") );
2889
2890 // increment sensor fields counter
2891 numFields++;
2892
2893 // set sensor fields counter
2894 buffer[4]=numFields;
2895 }
2896 else
2897 {
2898 // check if the data input type corresponds to the sensor
2899 if( checkFields(type, TYPE_FLOAT, 3) == -1 ) return -1;
2900
2901 // set data bytes (in this case, double is 4...)
2902 char valB1[4]; char valB2[4]; char valB3[4];
2903 memcpy(valB1,&val1,4);
2904 memcpy(valB2,&val2,4);
2905 memcpy(valB3,&val3,4);
2906 /*
2907 union {
2908     float f;
2909     char b[4];
2910 } u;
2911 u.b[3] = val[3];
2912 u.b[2] = val[2];
2913 u.b[1] = val[1];
2914 u.b[0] = val[0];
2915
2916 USB.println(u.f);
2917 */
2918 // check if new sensor value fits /1+4+4+4/
2919 if(!checkLength(13))
2920 {
2921     return -1;
2922 }
2923
2924 // concatenate sensor name to frame string
2925
2926 buffer[length-13] = (char)type;
2927 buffer[length-12] = valB1[0];
2928 buffer[length-11] = valB1[1];
2929 buffer[length-10] = valB1[2];
2930 buffer[length-9] = valB1[3];
2931 buffer[length-8] = valB2[0];
2932 buffer[length-7] = valB2[1];
2933 buffer[length-6] = valB2[2];
2934 buffer[length-5] = valB2[3];
2935 buffer[length-4] = valB3[0];
2936 buffer[length-3] = valB3[1];
2937 buffer[length-2] = valB3[2];
2938 buffer[length-1] = valB3[3];
2939 buffer[length] = '\\0';
2940
2941 // increment sensor fields counter
2942 numFields++;
2943 // update number of bytes field
2944 buffer[4] = frame.length-5;
2945

```

```

2946     }
2947
2948     return length;
2949 }
2950
2951 // FRAME 4 creat Rubèn serveix per enviar x valors // SMART CITIES
2952
2953 int8_t WaspFrame::addSensor(uint8_t type, double val1, double val2, double val3, int
val4)
2954 {
2955
2956     char str1[20];
2957     char str2[20];
2958     char str3[20];
2959     char str4[20];
2960
2961
2962
2963     if(_mode == ASCII)
2964     {
2965         // get name of sensor from table
2966         char numDecimals;
2967         numDecimals =(uint8_t)pgm_read_word(&(DECIMAL_TABLE[type]));
2968
2969         // convert from float to string
2970         dtostrf( val1, numDecimals, numDecimals, str1 );
2971         dtostrf( val2, numDecimals, numDecimals, str2 );
2972         dtostrf( val3, numDecimals, numDecimals, str3 );
2973         itoa( val4, str4, 10);
2974
2975
2976         // get name of sensor from table
2977         char name[10];
2978         strcpy_P(name, (char*)pgm_read_word(&(SENSOR_TABLE[type])));
2979
2980         // check if new sensor value fits in the frame or not
2981         // in the case the maximum length is reached, exit with error
2982         // if not, then add the new sensor length to the total length
2983         if(!checkLength( strlen(name) +
2984                         strlen(str1) + strlen(str2) + strlen(str3) +
2985                         strlen(str4) +
2986                         strlen(";") + strlen(";") +
2987                         strlen(";") + strlen(":") +
2988                         strlen("#")    ))
2989         {
2990             return -1;
2991         }
2992
2993
2994         int index_1 =
length-strlen(name)-strlen(":")-strlen(str1)-strlen(";")-strlen(str2)-strlen("
;")-strlen(str3)-strlen(";")-strlen(str4)-strlen("#");
2995         int index_2 =
length-strlen(":")-strlen(str1)-strlen(";")-strlen(str2)-strlen(";")-strlen(st
r3)-strlen(";")-strlen(str4)-strlen("#");
2996         int index_3 =
length-strlen(str1)-strlen(";")-strlen(str2)-strlen(";")-strlen(str3)-strlen("
;")-strlen(str4)-strlen("#");
2997         int index_4 =
length-strlen(";")-strlen(str2)-strlen(";")-strlen(str3)-strlen(";")-strlen(st
r4)-strlen("#");
2998         int index_5 =
length-strlen(str2)-strlen(";")-strlen(str3)-strlen(";")-strlen(str4)-strlen("
#");
2999         int index_6 =
length-strlen(";")-strlen(str3)-strlen(";")-strlen(str4)-strlen("#");
3000         int index_7 = length-strlen(str3)-strlen(";")-strlen(str4)-strlen("#");
3001         int index_8 = length-strlen(";")-strlen(str4)-strlen("#");
3002         int index_9 = length-strlen(str4)-strlen("#");
3003         int index_10 = length-strlen("#");
3004
3005
3006

```

```

3007 // add sensor tag
3008 memcpy ( &buffer[index_1], name, strlen(name) );
3009
3010 // add ':'
3011 memcpy ( &buffer[index_2], ":", strlen(":") );
3012
3013 // add input string defined in 'str1'
3014 memcpy ( &buffer[index_3], str1, strlen(str1) );
3015
3016 // add ';'
3017 memcpy ( &buffer[index_4], ";", strlen(";") );
3018
3019 // add input string defined in 'str2'
3020 memcpy ( &buffer[index_5], str2, strlen(str2) );
3021
3022 // add ';'
3023 memcpy ( &buffer[index_6], ";", strlen(";") );
3024
3025 // add input string defined in 'str3'
3026 memcpy ( &buffer[index_7], str3, strlen(str3) );
3027
3028 // add ';'
3029 memcpy ( &buffer[index_8], ";", strlen(";") );
3030
3031 // add input string defined in 'str3'
3032 memcpy ( &buffer[index_9], str4, strlen(str4) );
3033
3034
3035 // add separator '#'
3036 memcpy ( &buffer[index_10], "#", strlen("#") );
3037
3038 // increment sensor fields counter
3039 numFields++;
3040
3041 // set sensor fields counter
3042 buffer[4]=numFields;
3043 }
3044 else
3045 {
3046 // check if the data input type corresponds to the sensor
3047 if( checkFields(type, TYPE_FLOAT, 3) == -1 ) return -1;
3048
3049 // set data bytes (in this case, double is 4...)
3050 char valB1[4]; char valB2[4]; char valB3[4];
3051 memcpy(valB1,&val1,4);
3052 memcpy(valB2,&val2,4);
3053 memcpy(valB3,&val3,4);
3054 /*
3055 union {
3056     float f;
3057     char b[4];
3058 } u;
3059 u.b[3] = val[3];
3060 u.b[2] = val[2];
3061 u.b[1] = val[1];
3062 u.b[0] = val[0];
3063
3064 USB.println(u.f);
3065 */
3066 // check if new sensor value fits /1+4+4+4/
3067 if(!checkLength(13))
3068 {
3069     return -1;
3070 }
3071
3072 // concatenate sensor name to frame string
3073
3074 buffer[length-13] = (char)type;
3075 buffer[length-12] = valB1[0];
3076 buffer[length-11] = valB1[1];
3077 buffer[length-10] = valB1[2];
3078 buffer[length-9] = valB1[3];
3079 buffer[length-8] = valB2[0];

```

```

3080     buffer[length-7] = valB2[1];
3081     buffer[length-6] = valB2[2];
3082     buffer[length-5] = valB2[3];
3083     buffer[length-4] = valB3[0];
3084     buffer[length-3] = valB3[1];
3085     buffer[length-2] = valB3[2];
3086     buffer[length-1] = valB3[3];
3087     buffer[length] = '\\0';
3088
3089     // increment sensor fields counter
3090     numFields++;
3091     // update number of bytes field
3092     buffer[4] = frame.length-5;
3093
3094 }
3095
3096     return length;
3097 }
3098 //FRAME 1 Creat per enviar les dades de GASOS
3099
3100 int8_t WaspFrame::addSensor(uint8_t type, double val1, double val2, double
val3, double val4, double val5, int val6)
3101 {
3102
3103     char str1[20];
3104     char str2[20];
3105     char str3[20];
3106     char str4[20];
3107     char str5[20];
3108     char str6[20];
3109
3110
3111     if(_mode == ASCII)
3112     {
3113         // get name of sensor from table
3114         char numDecimals;
3115         numDecimals =(uint8_t)pgm_read_word(&(DECIMAL_TABLE[type]));
3116
3117         // convert from float to string
3118         dtostrf( val1, numDecimals, numDecimals, str1 );
3119         dtostrf( val2, numDecimals, numDecimals, str2 );
3120         dtostrf( val3, numDecimals, numDecimals, str3 );
3121         dtostrf( val4, numDecimals, numDecimals, str4 );
3122         dtostrf( val5, numDecimals, numDecimals, str5 );
3123         //itoa( val3, str3, 10);
3124         //itoa( val4, str4, 10);
3125         //itoa( val5, str5, 10);
3126         itoa( val6, str6, 10);
3127
3128
3129         // get name of sensor from table
3130         char name[10];
3131         strcpy_P(name, (char*)pgm_read_word(&(SENSOR_TABLE[type])));
3132
3133         // check if new sensor value fits in the frame or not
3134         // in the case the maximum length is reached, exit with error
3135         // if not, then add the new sensor length to the total length
3136         if(!checkLength( strlen(name) +
3137                         strlen(str1) + strlen(str2) + strlen(str3) +
3138                         strlen(str4) + strlen(str5) + strlen(str6) +
3139
3140                         strlen(";") + strlen(";") + strlen(";") +
3141                         strlen(";") + strlen(";")+
3142                         strlen(":") +
3143                         strlen("#")      ))
3144         {
3145             return -1;
3146         }
3147
3148
3149
3150         int index_1 =
length-strlen(name)-strlen(":")-strlen(str1)-strlen(";")-strlen(str2)-strlen("

```

```

;")-strlen(str3)-strlen(";")-strlen(str4)-strlen(";")-strlen(str5)-strlen(";")
-strlen(str6)-strlen("#");
3151 int index_2 =
length-strlen(":")-strlen(str1)-strlen(";")-strlen(str2)-strlen(";")-strlen(st
r3)-strlen(";")-strlen(str4)-strlen(";")-strlen(str5)-strlen(";")-strlen(str6)
-strlen("#");
3152 int index_3 =
length-strlen(str1)-strlen(";")-strlen(str2)-strlen(";")-strlen(str3)-strlen("
;")-strlen(str4)-strlen(";")-strlen(str5)-strlen(";")-strlen(str6)-strlen("#")
;
3153 int index_4 =
length-strlen(";")-strlen(str2)-strlen(";")-strlen(str3)-strlen(";")-strlen(st
r4)-strlen(";")-strlen(str5)-strlen(";")-strlen(str6)-strlen("#");
3154 int index_5 =
length-strlen(str2)-strlen(";")-strlen(str3)-strlen(";")-strlen(str4)-strlen("
;")-strlen(str5)-strlen(";")-strlen(str6)-strlen("#");
3155 int index_6 =
length-strlen(";")-strlen(str3)-strlen(";")-strlen(str4)-strlen(";")-strlen(st
r5)-strlen(";")-strlen(str6)-strlen("#");
3156 int index_7 =
length-strlen(str3)-strlen(";")-strlen(str4)-strlen(";")-strlen(str5)-strlen("
;")-strlen(str6)-strlen("#");
3157 int index_8 =
length-strlen(";")-strlen(str4)-strlen(";")-strlen(str5)-strlen(";")-strlen(st
r6)-strlen("#");
3158 int index_9 =
length-strlen(str4)-strlen(";")-strlen(str5)-strlen(";")-strlen(str6)-strlen("
#");
3159 int index_10 =
length-strlen(";")-strlen(str5)-strlen(";")-strlen(str6)-strlen("#");
3160 int index_11 = length-strlen(str5)-strlen(";")-strlen(str6)-strlen("#");
3161 int index_12 = length-strlen(";")-strlen(str6)-strlen("#");
3162 int index_13 = length-strlen(str6)-strlen("#");
3163 int index_14 = length-strlen("#");
3164
3165
3166
3167 // add sensor tag
3168 memcpy ( &buffer[index_1], name, strlen(name) );
3169
3170 // add ':'
3171 memcpy ( &buffer[index_2], ":", strlen(":") );
3172
3173 // add input string defined in 'str1'
3174 memcpy ( &buffer[index_3], str1, strlen(str1) );
3175
3176 // add ';'
3177 memcpy ( &buffer[index_4], ";", strlen(";") );
3178
3179 // add input string defined in 'str2'
3180 memcpy ( &buffer[index_5], str2, strlen(str2) );
3181
3182 // add ';'
3183 memcpy ( &buffer[index_6], ";", strlen(";") );
3184
3185 // add input string defined in 'str3'
3186 memcpy ( &buffer[index_7], str3, strlen(str3) );
3187
3188 // add ';'
3189 memcpy ( &buffer[index_8], ";", strlen(";") );
3190
3191 // add input string defined in 'str3'
3192 memcpy ( &buffer[index_9], str4, strlen(str4) );
3193
3194 // add ';'
3195 memcpy ( &buffer[index_10], ";", strlen(";") );
3196
3197 // add input string defined in 'str3'
3198 memcpy ( &buffer[index_11], str5, strlen(str5) );
3199
3200 // add ';'
3201 memcpy ( &buffer[index_12], ";", strlen(";") );
3202

```

```

3203         // add input string defined in 'str3'
3204         memcpy ( &buffer[index_13], str6, strlen(str6) );
3205
3206
3207         // add separator '#'
3208         memcpy ( &buffer[index_14], "#", strlen("#") );
3209
3210         // increment sensor fields counter
3211         numFields++;
3212
3213         // set sensor fields counter
3214         buffer[4]=numFields;
3215     }
3216     else
3217     {
3218         // check if the data input type corresponds to the sensor
3219         if( checkFields(type, TYPE_FLOAT, 3) == -1 ) return -1;
3220
3221         // set data bytes (in this case, double is 4...)
3222         char valB1[4]; char valB2[4]; char valB3[4];
3223         memcpy(valB1,&val1,4);
3224         memcpy(valB2,&val2,4);
3225         memcpy(valB3,&val3,4);
3226         /*
3227         union {
3228             float f;
3229             char b[4];
3230         } u;
3231         u.b[3] = val[3];
3232         u.b[2] = val[2];
3233         u.b[1] = val[1];
3234         u.b[0] = val[0];
3235
3236         USB.println(u.f);
3237         */
3238         // check if new sensor value fits /1+4+4+4/
3239         if(!checkLength(13))
3240         {
3241             return -1;
3242         }
3243
3244         // concatenate sensor name to frame string
3245
3246         buffer[length-13] = (char)type;
3247         buffer[length-12] = valB1[0];
3248         buffer[length-11] = valB1[1];
3249         buffer[length-10] = valB1[2];
3250         buffer[length-9] = valB1[3];
3251         buffer[length-8] = valB2[0];
3252         buffer[length-7] = valB2[1];
3253         buffer[length-6] = valB2[2];
3254         buffer[length-5] = valB2[3];
3255         buffer[length-4] = valB3[0];
3256         buffer[length-3] = valB3[1];
3257         buffer[length-2] = valB3[2];
3258         buffer[length-1] = valB3[3];
3259         buffer[length] = '\0';
3260
3261         // increment sensor fields counter
3262         numFields++;
3263         // update number of bytes field
3264         buffer[4] = frame.length-5;
3265
3266     }
3267
3268     return length;
3269 }
3270
3271 /*
3272 * checkFields(type, typeVal, fields ) - check type of value given by the user
3273 *
3274 * Parameters:
3275 * 'type' is the index given to the sensor

```



```

3276 * 'typeVal' is the type of variable given [0:uint8][1:int][2:double][3:char*]
3277 * 'fields' is the number of fields the sensor must have
3278 * All parameters given are checked by this function in order to see if the
3279 * inputs are OK
3280 *
3281 */
3282 int8_t WaspFrame::checkFields(uint8_t type, uint8_t typeVal, uint8_t fields)
3283 {
3284     uint8_t config;
3285     uint8_t nfields;
3286
3287     // *1* check sensor typeVal
3288
3289     config =(uint8_t)pgm_read_word(&(SENSOR_TYPE_TABLE[type]));
3290
3291     // special case (0 might be 1)
3292     if (config ==1)
3293     {
3294         if ((typeVal == 0)|| (typeVal == 1))
3295         {
3296             //OK
3297         }
3298         else
3299         {
3300             USB.print(F("ERR sensor type & value mismatch, "));
3301             USB.print( config, DEC);
3302             USB.print(F(" vs "));
3303             USB.println( typeVal, DEC);
3304             return -1;
3305         }
3306     }
3307     else
3308     {
3309         if (config == typeVal)
3310         {
3311             //OK
3312         }
3313         else
3314         {
3315             USB.print(F("ERR sensor type & value mismatch, "));
3316             USB.print( config, DEC);
3317             USB.print(F(" vs "));
3318             USB.println( typeVal, DEC);
3319             return -1;
3320         }
3321     }
3322
3323     // *2* check sensor number of fields
3324
3325     nfields =(uint8_t)pgm_read_word(&(SENSOR_FIELD_TABLE[type]));
3326     if (nfields == fields)
3327     {
3328         //OK
3329     }
3330     else
3331     {
3332         USB.println(F("ERR sensor type & number of fields mismatch"));
3333         return -1;
3334     }
3335
3336     return 1;
3337 }
3338
3339
3340
3341
3342 /// Private Methods //////////////////////////////////////
3343
3344 /*
3345 * checkLength (sum) - check if maximum length is reached
3346 *
3347 *
3348 * Return:

```

```

3349  * '1' : if new sensor value fits in the frame
3350  * '0' : if new sensor value does not fit in the frame
3351  *
3352  */
3353 uint8_t WaspFrame::checkLength(int sum)
3354 {
3355     if( length + sum <= _maxSize )
3356     {
3357         // if OK, add new length to total length
3358         length = length + sum;
3359         return 1;
3360     }
3361     else
3362     {
3363         // error
3364         return 0;
3365     }
3366 }
3367
3368
3369
3370
3371 /*
3372  * setID (moteID) - store mote ID to EEPROM
3373  *
3374  * This function stores the mote Identifier in EEPROM[147-162] addresses
3375  */
3376 void WaspFrame::setID(char* moteID)
3377 {
3378
3379     // set zeros in EEPROM addresses
3380     for( int i=0 ; i<16 ; i++ )
3381     {
3382         eeprom_write_byte((unsigned char *) i+MOTEDID_ADDR, 0x00);
3383     }
3384
3385     // clear the waspmote ID attribute
3386     memset( _waspmoteID, 0x00, sizeof(_waspmoteID) );
3387
3388     // set the mote ID from EEPROM memory
3389     for( int i=0 ; i<16 ; i++ )
3390     {
3391         if( moteID[i] != '#' )
3392         {
3393             eeprom_write_byte((unsigned char *) i+MOTEDID_ADDR, moteID[i]);
3394             _waspmoteID[i] = moteID[i];
3395         }
3396         else
3397         {
3398             eeprom_write_byte((unsigned char *) i+MOTEDID_ADDR, '_');
3399             _waspmoteID[i] = '_';
3400         }
3401
3402         // break if end of string
3403         if( moteID[i] == '\0' )
3404         {
3405             break;
3406         }
3407     }
3408 }
3409
3410
3411
3412 /*
3413  * getID (moteID) - read mote ID from EEPROM
3414  *
3415  * This function reads the moteID previously stored in EEPROM[147-162] addresses
3416  */
3417 void WaspFrame::getID(char* moteID)
3418 {
3419     // clear the waspmote ID attribute
3420     memset( _waspmoteID, 0x00, sizeof(_waspmoteID) );
3421

```

```

3422     // read mote ID from EEPROM memory
3423     for(int i=0 ; i<16 ; i++ )
3424     {
3425         _wasmoteID[i] = Utils.readEEPROM(i+MOTEID_ADDR);
3426         moteID[i] = _wasmoteID[i];
3427     }
3428     moteID[16]='\0';
3429
3430 }
3431
3432
3433 /*
3434  * storeSequence (seqNumber) - store the sequence number to EEPROM[163] address
3435  *
3436  */
3437 void WaspFrame::storeSequence(uint8_t seqNumber)
3438 {
3439     // store frame sequence number to EEPROM[163]
3440     eeprom_write_byte((unsigned char *) SEQUENCE_ADDR, seqNumber);
3441
3442 }
3443
3444
3445
3446
3447 /*
3448  * readSequence (void) - read the sequence number from EEPROM[163] address
3449  *
3450  */
3451 uint8_t WaspFrame::readSequence(void)
3452 {
3453     // read frame sequence number from EEPROM[163]
3454     return Utils.readEEPROM(SEQUENCE_ADDR);
3455 }
3456
3457
3458
3459
3460
3461 /*
3462  * decrementSequence (void) - decrement the sequence number
3463  *
3464  * This function is useful for external function which only need to send a
3465  * Frame with no sensor but some kind of feature. i.e. time sync frame
3466  */
3467 void WaspFrame::decrementSequence(void)
3468 {
3469     uint8_t aux_sequence = readSequence();
3470
3471     aux_sequence--;
3472
3473     // store the new sequence number
3474     storeSequence(aux_sequence);
3475 }
3476
3477 /*
3478  * addTimestamp () - add time stamp to the frame from RTC
3479  *
3480  *
3481  * Return:
3482  * '1' : if new sensor value does not fit in the frame
3483  * '0' : if new sensor value fits in the frame
3484  *
3485  */
3486
3487 int8_t WaspFrame::addTimestamp()
3488 {
3489     //ensure RTC is ON
3490     RTC.ON();
3491     unsigned long epochTime = RTC.getEpochTime();
3492
3493     if (_mode == ASCII)
3494     {

```

```

3495         return addSensor(SENSOR_TST, epochTime);
3496     }
3497     else
3498     {
3499         // check if new sensor value fits /1+4/
3500         if(!checkLength(5))
3501         {
3502             return -1;
3503         }
3504
3505         // look for the separator
3506         uint8_t* pointer;
3507         pointer = (uint8_t*) memchr(&buffer[9], '#', length-5-9);
3508
3509         if (pointer == NULL)
3510         {
3511             return -1;
3512         }
3513
3514         // increment index to reach beginning of first payload byte
3515         pointer++;
3516         pointer++;
3517
3518         // calculate total buffer size to move
3519         uint32_t size = length - 5 - (pointer-&buffer[0]);
3520
3521         // move data to make room for timestamp
3522         memmove(pointer+5, pointer, size);
3523
3524         // copy timestamp to buffer
3525         char type = SENSOR_TST;
3526         memcpy(pointer,&type,1);
3527         memcpy(pointer+1,&epochTime,4);
3528
3529         // increment sensor fields counter
3530         numFields++;
3531         // update number of bytes field
3532         buffer[4] = frame.length-5;
3533
3534         return length;
3535     }
3536 }
3537
3538
3539 /// Preinstantiate Objects //////////////////////////////////////
3540
3541 WaspFrame frame = WaspFrame();
3542
3543

```