Smart Sensor Development Kit

User Manual

Version date 16/06/2021

Development kit for the DLL for the *Smart Sensor*

Version 1.0

Index

1.License agreement

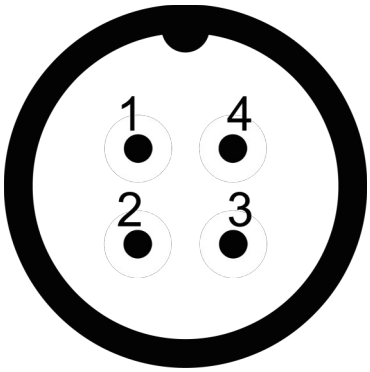
Read this license agreement thoroughly before using the System. Using this System is subject to the acceptance of this agreement.
If you choose to refuse the following conditions, please return this System to the point of purchase for a complete refund. This agreement involves Tecnosoft srl, Peschiera Borromeo, Milano, Italy (henceforth called Tecnosoft) and the User (be it a physical or juridical person) for the following Smart Sensor Development kit (henceforth called Software): "Smart Sensor".
Tecnosoft grants the user a non-exclusive right to use a copy of the software on a single computer provided that the user accepts the following conditions.

1. **User license**. *The Smart Sensor Development kit is property of Tecnosoft and cannot be copied nor sold without the prior written authorization of Tecnosoft. The Smart Sensor Development kit is protected by Italian and European Laws and by International Treaties concerning intellectual properties.*
2. **Exclusion of liabilities**. *Except for what stated by applicable laws, in no case can Tecnosoft be considered liable for damages or losses, direct or indirect, including, but not limited to, loss or missing income, suspension of activities, loss of information or any other monetary or economical damage, deriving from proper or improper use of the Smart Sensor Development kit even if Tecnosoft has been advised of the possibility of such damages. In any case, the responsibility of Tecnosoft for such damages will be limited to the price paid for the Smart Sensor Development kit. This clause is applied even if the User does not accept the Smart Sensor Development kit.*
3. **Use of Firmware results**. *It is User's responsibility to check that results given by the Smart Sensor Development kit are correct and appropriate. In no case the Smart Sensor Development kit should be used if such use can be threatening to the health or life of human beings. This clause is applied even if the User does not accept the Software.*
4. **Updates**. *If the Smart Sensor Development kit is an update of a previous version, the license is transferred from the old version to the update. Only the update can be used, unless the update is destroyed.*
5. **Separation of components**. *The Smart Sensor Development kit is licensed as a single product. Components cannot be separated.*
6. **Limitations**. *The User cannot convert, decode, decompile or disassemble the Smart Sensor Development kit, except for what is explicitly requested by applicable laws.*

2. Communication between node and transducer

Communication between node and transducer is performed at TTL levels (3.3V or 5V depending on supply type) at 9600 baud, 8 bit data, 1 stop, no parity.

The IT uses the connector below:



Pin description (male used on nodes):

1. TX: From SS to node (white)
2. RX: From SS to node (black)
3. Ground (green)
4. Supply (3.3V or 5.0V) (red)

Communication occurs in frames. A frame is started by a 0xFF character and the number of characters in the frame is calculated using the data length field of the frame.
A frame is not be terminated by a 0xFF character. When the needed number of characters is received,, the frame is implicitly terminated. A 0xFF character received before the packet is completely received will abort reception and start reception of a new frame.
Within frames bytes 0xFE and 0xFF are encoded as 0xFE followed by <aabbccdd> where aa=01 encodes a 0xFE and aa=10 encodes a 0xFF; an aa value of 00 or 11 means nothing is encoded.
So 0xFE + 0b00000110 means 0xFE followed by 0xFF.
A packet is encoded as follows:

```
typedef struct { unsigned
      char dest; unsigned
      char source; unsigned
      char type; unsigned
      char filler; unsigned
      size; unsigned
      sequence;
      union { unsigned char
            content[NET_MAX_CONTENT];
      };
} NET_PACKET;
```

Note: unsigned char are eight bits and unsigned are 16 bits little endian.
Dest is the destination address for the packet, source is the address of the unit generating the packet. These may be a value in range 0x01 to 0xFE or be one of the special values:

```
#define NET_MASTER_ID 0xFF
#define NET_ALL_ID 0x00
```

Type is the packet type; types 0x00 to 0x7F have the same meaning for all Smart Sensor (e.g. READ), types 0x80 to 0xFF are transducer dependent: it is not safe to issue commands in this range to a transducer if the transducer type and the meaning of the packet for that transducer is known.

Filler is just that: a dummy field used to keep the following fields word aligned when the structure is stored in memory (this is necessary for some microprocessors like the MSP430).
Size is the number of bytes following the sequence field.
Sequence is used to distinguish different packets when sending them on a media that uses streaming to improve communication performance.
Normally an unit will communicate with a transducer in a ping pong way.
Another use of the sequence field is when sending data that is serialized (like reading block 0, 1 and so on of memory).

Standard packets (in range 0x00 to 0x7F)

Three standard packets are currently defined:

Type	*Name*	*Description*
0x00	*NET_UNIT*	*Query information about the connected Smart Sensor*
0x01	*NET_CHANNEL*	*Query information about specific channel*
0x02	*NET_READ*	*Read value a channel*

NET_UNIT Packet

This packet has the following content.

```
typedef struct {
   // The following fields are only in answer message
unsigned char identity[8];
   unsigned      model;      // Model of Smart Sensor unsigned
numberOfChannels;
   unsigned long calibration;// Calibration date (in seconds since 1/1/2000)
unsigned long expiry;     // Expiration date (in seconds since 1/1/2000)
} NET_UNIT_PACKET;
```

The packet is sent to the transducer with size==0 and the above data is returned.
The model field identifies the specific transducer type: a program may use this field to understand the type of the transducer and later use packets with type in range 0x80 to 0xFF for the specific transducer.
Identity is unique: no two transducers will ever be manufactured with the same identity.
A unit is expected to get this packet from the transducer to verify the transducer type and validity. Then the unit may issue NET_CHANNEL packets for channels 0 to numberOfChannels-1.

NET_CHANNEL Packet

This packet has the following content:

```
typedef struct {
   unsigned      channel;    // Channel number (0..<numberOfChannels-1)

   // The following fields are only in answer message
   unsigned      type;       // Type of transducer (used e.g. to interpret
                         CHANNEL_FAILURE error codes)    unsigned      supply;
//  Current  needed  for  measure  in  milliamps       unsigned  char  unit[16];
// Label for unit type (e.g. Pa)
    unsigned char measure;    // Type of unit of measure (see above)
    unsigned char radians;    // (2*<exponent of >)+128;
    unsigned char steradians; // (2*<exponent of >)+128;
    unsigned char meters;     // (2*<exponent of >)+128;
    unsigned char kilograms;  // (2*<exponent of >)+128;
    unsigned char seconds;    // (2*<exponent of >)+128;
    unsigned char amperes;    // (2*<exponent of >)+128;
```

```
    unsigned char kelvins;    // (2*<exponent of >)+128;
    unsigned char moles;      // (2*<exponent of >)+128;
    unsigned char candelas;   // (2*<exponent of >)+128;
} NET_CHANNEL_PACKET;
```

This packet is issued to the Smart Sensor with only the channel field filled in (size==2). The answer is a packet filled with all the above information.
The type of the transducer may be useful to give a more complete interpretation of possible error conditions, but it is by no means required that a reading unit interprets it.

A unit is expected to read the supply field and evaluate if the current required for the measurement can currently be sourced or delay the measure if this is not possible.

The unit field is assumed to match the unit definition given with the exponents(radians...) and the measure fields: this is assumed to be user readable.

The measure constant is one of the following values:

```
// 0: Unit is described by the product ofsi base units, plus radians and
steradians,
// raised to the powers recorded in fields 2-10.
#define MEASURE_TYPE_SI 0
// 1: Unit is U/U, where U is described by the product of SI base units, plus
radians
// and steradians, raised to the powers recorded in fields 2-10.
#define MEASURE_TYPE_UU 1
// 2: Unit is log10(U), where U is described by the product of SI base units,
plus radians
// and steradians, raised to the powers recorded in fields 2-10.
#define MEASURE_TYPE_LOG10U 2
// 3: Unit is log10(U/U),whereU is described by the product of SI base units,
plus radians
// and steradians, raised to the powers recorded in fields 2-10.
#define MEASURE_TYPE_LOG10UU 3
// 4: The associated quantity is digital data (e.g., a bit vector) and has no
unit.
// Fields 2-10 shall be set to 128. The "digital data" type applies to data
that represents counting,
// such as the output of an ADC, or that represents a state, such as the
current positions
// of a bank of switches.
#define MEASURE_TYPE_DIGITAL 4
// 5: The associated physical quantity is represented by values on an arbitrary
scale
// (e.g., hardness). Fields 2-10 are reserved, and shall be set to 128
#define MEASURE_TYPE_ARBITRARY 5
```

For more information on this issue refer to IEEE Std 1451.2.1997.
Please note that the above definition of unit requires that the readouts are returned in MKSA units. Temperatures are hence measured in Kelvins. This is not a problem because the packet is not supposed to be human readable.

NET_READ Packet

This packet has the following content:

```
typedef struct
{    unsigned    channel;
unsigned      command;
```

```
   // The following fields are only in answer message
float         value;
unsigned      error;
} NET_READ_PACKET;
```

When issued to the Smart Sensor only the first two fields should be filled in (size==4), the channel being a value in range 0 to numberOfChannels-1 and the command being one of the following constants:

```
// No command: just sending message to query reading status
#define CHANNEL_COMMAND_NONE 0
// Start reading channel
#define CHANNEL_COMMAND_START 1
```

When reading a CHANNEL_COMMAND_START command would be issued followed by repeated CHANNEL_COMMAND_NONE commands, till a final result is returned.

A Smart Sensor will always answer with the full packet, the value may be NaN and the error field is defined by the following constants:

```
// Channel has been read and value is being returned
#define CHANNEL_READ_OK  0x0000
// Channel is in overflow condition
#define CHANNEL_OVERFLOW 0x0100
// Channel is in underflow condition
#define CHANNEL_UNDERFLOW 0x0200
// Readout not ready yet, but wait
#define CHANNEL_WAIT   0xFE00
// Failure reading channel (the low byte will contain additional information)
#define CHANNEL_FAILURE  0xFF00
```

The high byte of the returned constant is as defined above, when receiving a CHANNEL_WAIT response, the external unit is expected to query again.

The low byte of the error field depends on the sensor type and is expected to give additional information; for example 'broken wire' in a 4□20 mA transducer.