



**SKX OPEN**

**SKX ADVANCE**

**ZN1RX-SKXOPEN**



Edition 2  
Version 1.1

## INDEX

|   |    |
|---|----|
| 1. Introduction .....   | 3  |
| 1.1. SKX Interface .....  | 3  |
| 1.2. SKX Installation.....  | 5  |
| 1.3. SKX Advance: Application Program .....                         | 5  |
| 1.3.1. SKX Advance Basic Specifications .....                       | 6  |
| 1.4. Differences between SKX Open and SKX Advance applications..... | 6  |
| 1.5. Advantages of the product and the application.....             | 7  |
| 2. ETS Parameterization .....                                       | 8  |
| 2.1. Communication configuration .....                              | 8  |
| 2.2. Frame configuration .....                                      | 11 |
| 2.2.1. Frame definition .....                                       | 11 |
| 2.2.2. Special frames.....  | 12 |
| 2.2.3. Special characters .....                                     | 14 |
| 2.3. Parameters Groups.....   | 17 |
| 2.3.1. 1 bit objects .....  | 18 |
| 2.3.2. 1 byte Objects.....  | 20 |
| 2.3.3. 14 bytes Objects .....                                       | 26 |
| 2.4. Error objects.....   | 28 |
| 2.4.1. Error examples.....  | 29 |
| 2.5. Configuration examples.....                                    | 34 |
| 2.5.1. 1 bit objects .....  | 34 |
| 2.5.2. 1 byte objects .....   | 35 |
| 2.5.3. 14 bytes objects.....  | 38 |
| 3. Product summary.....   | 40 |

# 1. INTRODUCTION

## 1.1. SKX INTERFACE

**SKX** is a **ZENNIO** interface that allows the connection of the KNX bus with other devices through a **RS-232** serial and bidirectional communication.



Figure 1.1. SKX connection scheme

### Nomenclature

They are defined below the most frequently used terms throughout this manual:

- **SKX:** from now on, SKX will term the SKX Open interface, in order not to create misunderstandings between the hardware and the application program of the same name.
- **SKX Advance:** application program that can be downloaded on SKX and that allows managing the KNX – RS232 communication.
- **RS-232:** serial communication type.
- **Data Terminal Equipment:** external device that will be integrated through the RS-232 serial port.

Next, the most significant characteristics of the interface are shown, as well as a SKX elements scheme (figure 1.2):

- Reduced size: 45 x 45 x 14 mm.
- Several communication speeds and error detection mechanisms.
- Ideal for M2M applications.
- Data saving in case of bus power failure.

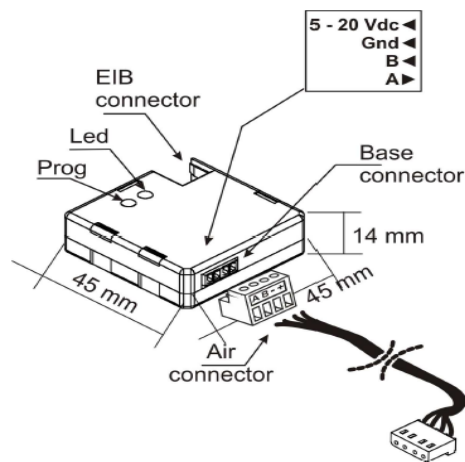


Figure 1.2. SKX. Elements scheme

The “**Prog**” button is used to set SKX in programming mode. If this button is held while plugging the device into the KNX bus, it goes into secure mode.

SKX has a **bicolour LED** on the front of the hardware. This LED has a double functionality: on the one hand, if it lights in red, it indicates that the interface is in the programming mode and if the LED blinks red every 0.5 seconds, the interface is in the secure mode. On the other hand, the LED can work as a **transmission indicator**. If it lights green, it indicates that data have been sent or received through the serial port, staying on 0.3 seconds every time a communication occurs.

SKX interface can be programmed with two different application programs: **SKX Open** and **SKX Advance**, which differences will be explained in the section 1.4.

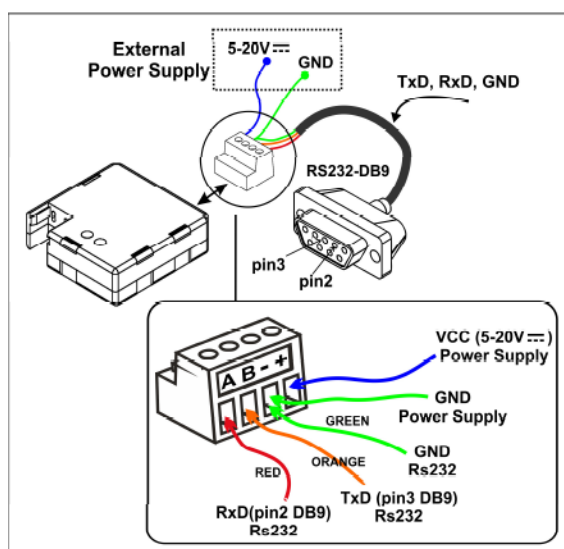
## 1.2. SKX INSTALLATION

SKX installation is very **easy**: just connect the interface to the KNX bus through its specific connector and SKX will be ready to be programmed.

SKX does not need external power supply different from the KNX bus one. However, the RS-232 does need to be powered independently from the KNX bus (this power is usually got from the terminal equipment connected to the serial bus).

The SKX – RS232 connection is made through a specific terminal block that eases its manipulation and installation.

In the following figure it can be seen a typical **SKX – RS232** connection:



| SKX TERMINAL BLOCK | RS-232 BUS |
|--------------------|------------|
| A                  | RSA        |
| B                  | RSB        |
| -                  | Ground     |
| +                  | +12V       |

Figure 1.3. SKX - RS-232 connection

## 1.3. SKX ADVANCE: APPLICATION PROGRAM

**SKX Advance** is an application that can be downloaded on the **SKX** interface. Its main function is to manage the communication between KNX and the RS-232 protocol, allowing the configuration of all the interchanged information between the bus and the terminal equipment. This information interchange is **bidirectional**, i.e., it is possible to send data from the KNX bus to the terminal equipment and vice versa. In the Figure 1.4 there is an example of this of communication.

SKX Advance is able to send or receive any kind of communication frame provided that it meets the criteria for RS-232 messages allowed (see section 2.2.1).

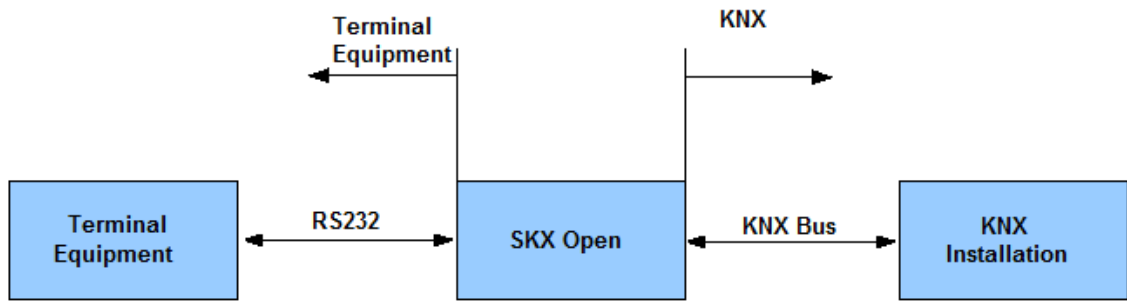


Figure 1.4. Bidirectional communication

### 1.3.1. SKX ADVANCE BASIC SPECIFICATIONS

It is shown below the basic characteristics of the SKX Advance application:

- **Velocity:** 1200, 1400, 4800, 9600 or 19200 bauds.
- **Parity:** even, odd, no parity
- **Reception complete mode:** Time Out, end-frame byte
- **Number of communication objects:** 65 (40 of 1 bit, 20 of 1 byte and 5 of 14 bytes)
- **Error identification:** several 1 bit objects
- **Protocol length:** the frames configured by parameter in SKX Advance may have a length greater than 25 bytes, thanks to the possibility of using special characters on their configuration. Altogether, **up to 29 bytes can be sent/received** for every message.

## 1.4. DIFFERENCES BETWEEN SKX OPEN AND SKX ADVANCE APPLICATIONS

SKX interface can be programmed with two different application programs: **SKX Open** and **SKX Advance**.

The main differences between the two application programs are shown in the next table:

|                               | SKX OPEN                               | SKX ADVANCE  |
|-------------------------------|--|--|
| <b>Number of Objects</b>      | 48                                     | 65   |
| <b>Object Type</b>            | 1 bit                                  | 1 bit<br>1 Byte<br>14 Bytes                                    |
| <b>Frame Type</b>             | Fixed                                  | Variables (depending on the object value)                      |
| <b>Frame Length</b>           | Up to 10 bytes                         | Up to 29 bytes   |
| <b>Frames Acknowledgement</b> | The whole message must match           | Just acknowledgement of indicated parts (omission of the rest) |
| <b>Checksum</b>               | No                                     | Yes  |
| <b>Confirmation (ACK)</b>     | No                                     | Yes  |
| <b>Configurability</b>        | - Communication<br>- Parameters Groups | - Communication<br>- Frames<br>- Parameters Groups             |

Table 1.1. Differences between SKX Open and SKX Advance

## 1.5. ADVANTAGES OF THE PRODUCT AND THE APPLICATION

The following advantages are associated to the fact of downloading the application program SKX Advance on the SKX interface:

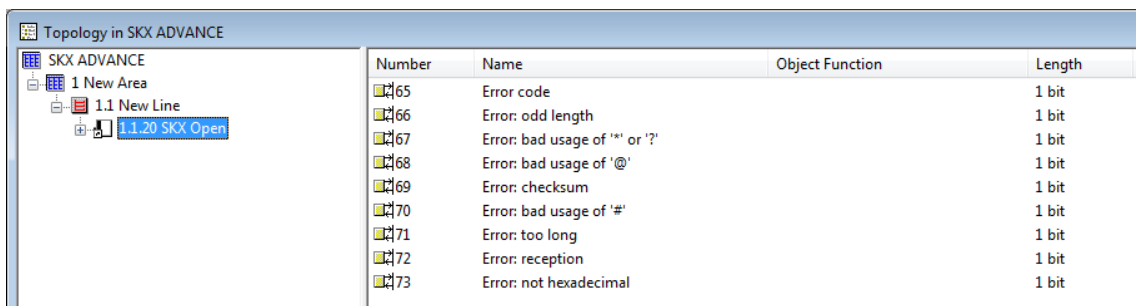
- Expansion of the installation. It allows **integrating** in a KNX installation other devices that do not have KNX communication but a RS-232 serial port.
- Adaptability. SKX Advance is able to perfectly adapt to the protocol that governs the functioning of the terminal equipment, regardless its complexity, i.e., SKX Advance **adapts to the terminal equipment** and not the other way.
- Communication versatility. SKX Advance has objects of different types (1 bit, 1 byte, 14 bytes) with which it interchange information with the terminal equipment in both directions (bidirectionality).
- Configurability. The messages of the RS-232 communication can be configured with all the normal characteristics of a serial communication, like the header and footer frames, checksum, ACK, etc., thus enabling the sending of dynamic messages of variable length.

## 2. ETS PARAMETERIZATION

With the SKX Advance application program is possible to integrate any terminal equipment with RS-232 interface in a KNX installation if you know the communication frames that the terminal equipment uses for every order.

SKX Advance has **65 communication objects of different sizes** with which it will be possible to interact with the KNX and RS-232 protocol. It has also several 1 bit objects to detect possible errors. Therefore, every defined frame has associated a communication object. All these objects will be explained in detail in the corresponding sections.

The configurable ETS parameters for SKX Advance are presented below.



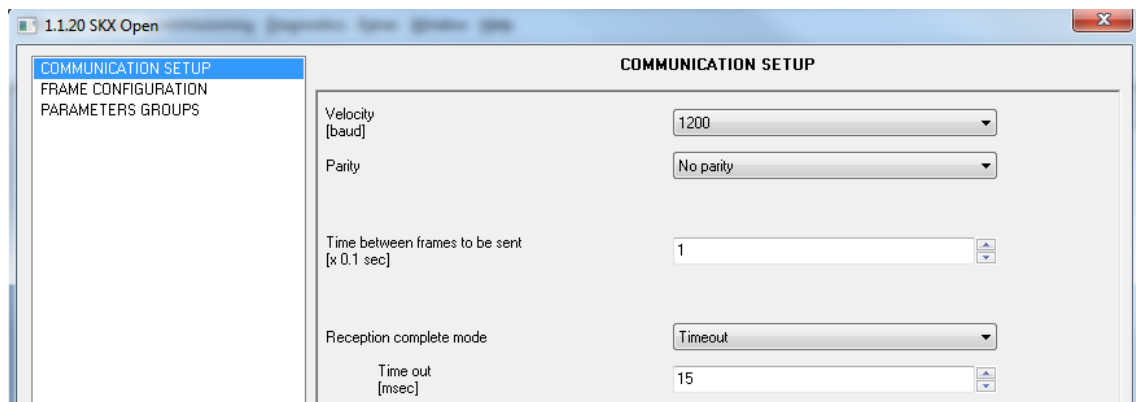
| Number | Name                           | Object Function | Length |
|--------|--------------------------------|-----------------|--------|
| 65     | Error code                     |                 | 1 bit  |
| 66     | Error: odd length              |                 | 1 bit  |
| 67     | Error: bad usage of '*' or '?' |                 | 1 bit  |
| 68     | Error: bad usage of '@'        |                 | 1 bit  |
| 69     | Error: checksum                |                 | 1 bit  |
| 70     | Error: bad usage of '#'        |                 | 1 bit  |
| 71     | Error: too long                |                 | 1 bit  |
| 72     | Error: reception               |                 | 1 bit  |
| 73     | Error: not hexadecimal         |                 | 1 bit  |

Figure 2.1. Default communications objects

Every configuration window of SKX Advance is explained next.

### 2.1. COMMUNICATION CONFIGURATION

The general configuration window that appears the first time “Edit Parameters” is clicked on is the following:



COMMUNICATION SETUP

Velocity [baud] 1200

Parity No parity

Time between frames to be sent [x 0.1 sec] 1

Reception complete mode Timeout

Time out [msec] 15

Figure 2.2. SKX Advance: Communication Setup



The following general parameters, related to the frame transmission, can be configured in this window:

- **Velocity (bauds):** 1200, 2400, 9600, 19200
- **Parity:** No parity, odd, even
- **Time between frames to be sent (tenths of second):** it is the minimum time between frames to be sent through the serial port. This parameter is used to separate consecutive frames that must be sent to the terminal equipment. This can be useful, for example, when linking more than one object to the same group address. Thus, SKX will be able to send that frames in an orderly way, enabling a perfect reception and interpretation of the frames on the terminal equipment. (**Note:** *take into account the terminal equipment characteristics when configuring the “Time between frames” parameter*).
- **Reception complete mode:** indicates the way SKX Advance detects when all the frame characters have been received. There are two ways to detect the end of frame:
  - **Time Out:** it is a minimum time (milliseconds) between frames. SKX Advance knows that a frame has been completely received once this time has elapsed after receiving the last bit of the frame.
  - **End-frame byte:** it is a byte with a specific value with an unambiguous interpretation, so that SKX Advance will know the frame reception has finished when it receives this byte. When selecting this option, a new drop.-down box will appear, to configure a security time out, defined as the maximum time SKX Advance waits to receive the end-frame byte.

**Note:** *before configuring a Time Out for the frame reception, please take into account the sending duration of every byte, shown in the next table:*

| Velocity (bauds) | Parity | Time/byte (ms) |
|------------------|--------|----------------|
| 1200             | Yes    | 9,167          |
|                  | No     | 8,333          |
| 2400             | Yes    | 4,583          |
|                  | No     | 4,167          |
| 4800             | Yes    | 2,292          |
|                  | No     | 2,083          |
| 9600             | Yes    | 1,146          |
|                  | No     | 1,042          |
| 19200            | Yes    | 0,573          |
|                  | No     | 0,521          |

If a Time Out of 3 milliseconds and a velocity of 2400 bauds are configured, SKX Advance will provoke a "Time Out error" for every entry datum, since 4,583 milliseconds are necessary to complete a byte transfer (if there is a bit of parity), value greater than the configured Time Out (3ms).

Note: this Time Out error is not indicated by none of the error identification objects.

It is convenient to read the following example to better understand the Time Out concept:

**Example:** one terminal equipment takes 80 ms to send its complete frame.

• **First Case (Time Out too long):** the parameterized Time Out is 30 ms. The terminal equipment sends a second frame just after the first one. The next figure shows this behaviour.

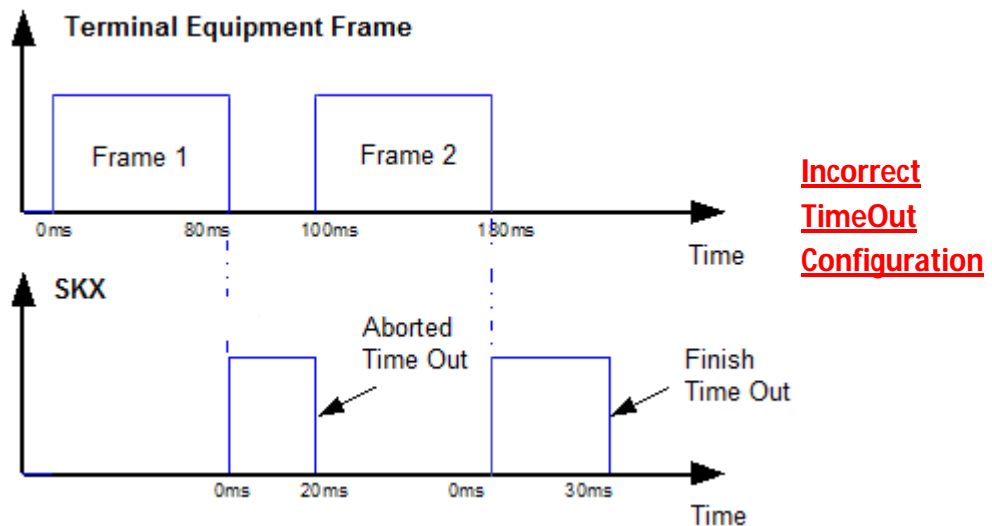


Figure 2.3. Time Out too long

At the end of the first frame, the Time Out started to count but another frame arrived before finishing the Time Out, so it was aborted and started to count again at the end of the second frame. In this case, as the Time Out comes to an end (30 ms), SKX Advance thinks that the frame has finished. But two frames have arrived until the end of frame has been detected, so SKX Advance considers the frame as unknown and it does not send anything.

• **Second Case (Time Out well defined):** the defined Time Out is now 10 ms. The next figure shows this behaviour:

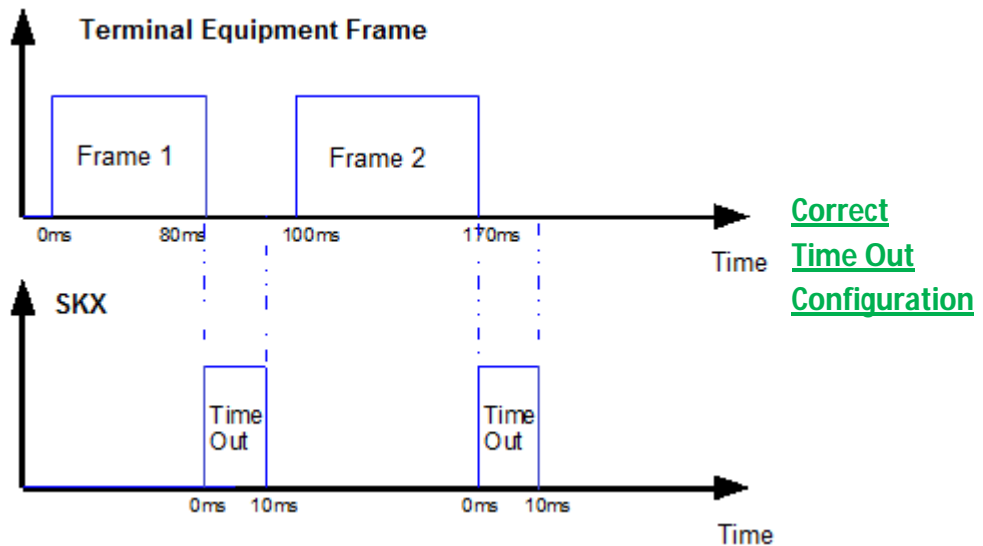


Figure 2.4. Time Out well defined

*In this case, the Time Out is well defined, so SKX Advance is able to recognize the two sent frames.*

So that, it is very **important to define correctly the Time Out**, taking into account that the time between sent frames from the terminal equipment, since as it was shown previously, a bad defined Time Out (too long or too short) could generate errors when receiving the messages.

## 2.2. FRAME CONFIGURATION

### 2.2.1. FRAME DEFINITION

The definition/parameterization of the communication frames is carried out by means of **hexadecimal characters** (2 characters for byte); therefore, only the characters between 0-9 and A-F are allowed to define a frame (excepting the special characters). It is **compulsory** that the characters A-F are **in capitals**.

**Note:** *It is convenient to know that a hexadecimal frame, for example “0x2B 0x7F 0x34” must be typed on ETS this way: “2B7F34”.*

When defining a serial port frame, ETS has a parameterization text box to type **up to 20 characters**. In order to define a frame with more characters, thus achieving a greater versatility in the communication (variable data, non fixed frame sizes, etc.) it is allowed to configure different sections of the frame. This way, it will possible to transmit/receive **up to 29-bytes-frames**.

Communication frames can be **odd**; what must necessarily be **even** is the set of characters that are typed by parameter (2 characters for byte).

The non-compliance of any of these frame parameterization requirements will make SKX Advance to send **error statuses** to the KNX bus after being programmed, through the communication objects enabled for that purpose and that will be explained later in this manual.

## 2.2.2. SPECIAL FRAMES

There is the possibility of configuring a set of special frames that allow a **complete communication** with the equipment that is going to be integrated in the KNX installation through RS-232.

The utilization of these frames is completely **optional**. The aim of them is to provide with enough mechanisms to generate dynamic messages in a communication protocol, enabling the use of headers, footers, subframes, and the sending of automated acknowledgements.

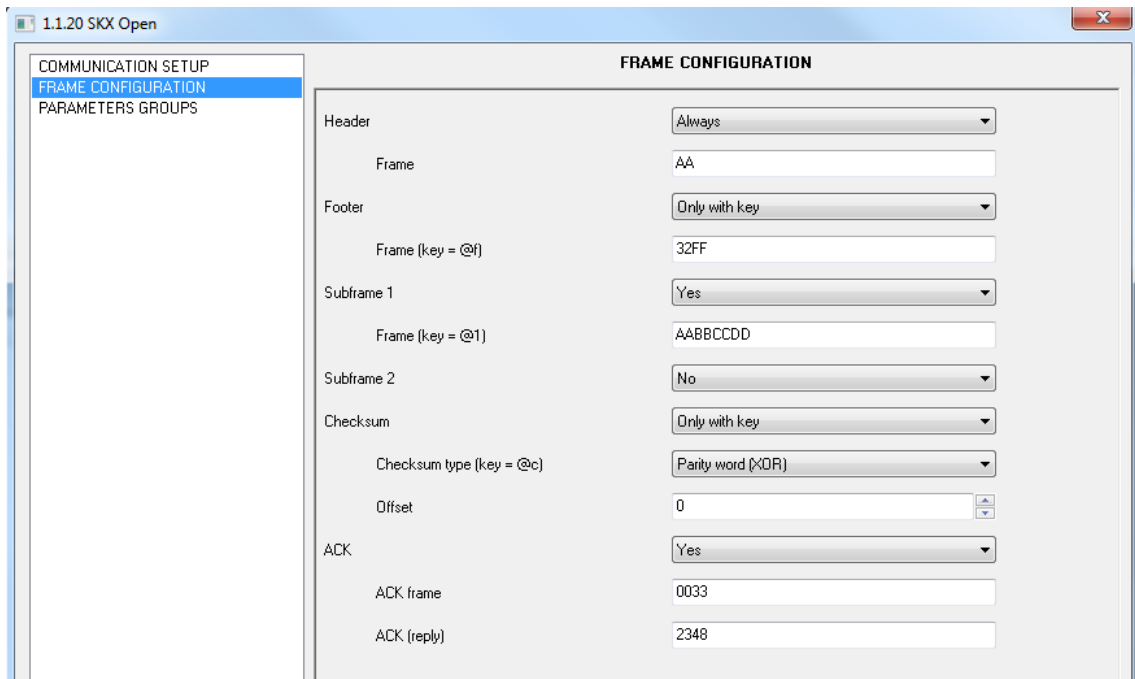


Figure 2.5. SKX Advance: Frame configuration (special frames)

Next, the explanation of each special frame:

- **Header:** this frame can appear at the beginning of every sent or received frame automatically (“Always”) or just when indicated in the frame (“Only with key”) by means of the special character ‘@h’. If “No” is chosen, the header will not be included in the frame.
- **Footer:** this special frame is included at the end of the defined frames (by parameter). Like header, its insertion can be disabled (“No”), or it can be always included at the end of the frame (“Always”) or only when indicated (“Only with key”) by means of the special character ‘@f’.

- **Subframes 1 and 2:** they can be included in any part of the defined frame, by means of the keys '@1' y '@2' respectively.
- **Checksum:** this special frame allows including a checksum, calculated in every case (reception or sending) from the first byte till the byte before the checksum one. It can be included in the frame by means of the key '@c' or it can be pointed out that every frame has a determined checksum at the end (after the footer, if any). There is also an additional parameter (Offset) that tells the program to calculate the checksum starting from the byte indicated in this parameter.

The supported checksum types are the following:

- **Parity word:** XOR operation, byte by byte.
  - **Modular sum:** it sums all the frame bytes, modulo 256.
  - **Modular sum complemented:** similar to the previous one, but in this case, the result is performed 2's complement.
  - **CRC-8 y CRC-16:** cyclic redundancy check of 8 and 16 bits respectively. It is necessary to write a decimal number to define the CRC characteristic polynomial.
- **Acknowledgements (ACK):** SKX Advance allows sending automatically fixed ACK frames to any frame received from the serial port. This frame is defined in the "ACK" field. The application program also allows configuring a specific frame (in the "ACK reply" field) as an acknowledgement sent by the terminal equipment, so when SKX Advance receives this frame, it will not send any kind of ACK. Therefore, it is necessary to define here the possible ACK frames that the terminal equipment sends.

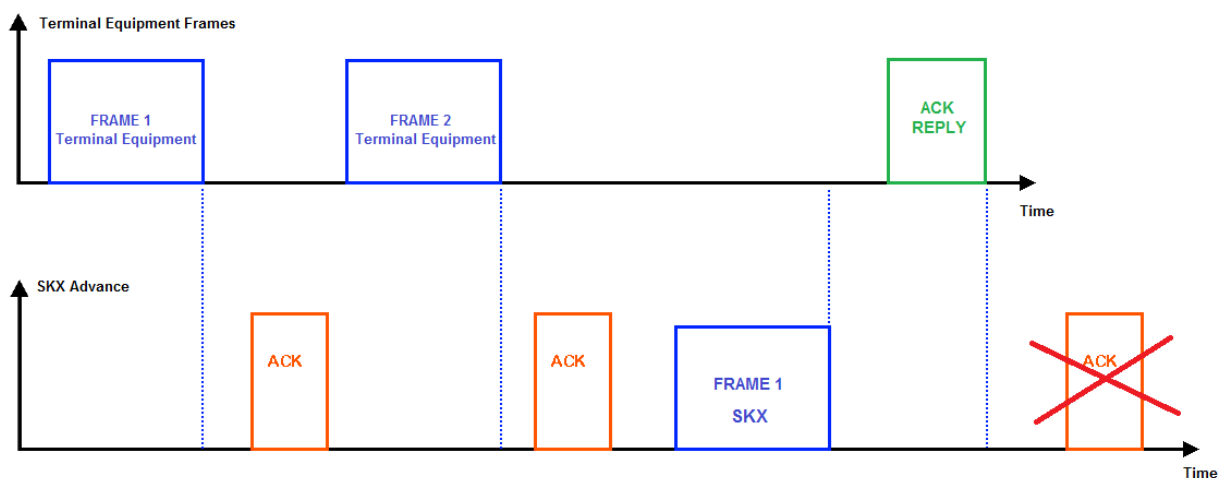



Figure 2.6. ACK frames and ACK reply

## 2.2.3. SPECIAL CHARACTERS

As mentioned before, there is a set of special characters that can be added to the defined frames to give them a greater versatility, thus enabling the generation and detection of a greater set of messages.

Besides, this type of characters intend to allow the utilization of regular expressions in the frame definition.

Two groups can be distinguished: special characters for reception and special characters for transmission and reception. Next, an explanation:

 **Special characters for reception:** these characters can only be used in the parameterization of frames that will be received through the serial port; if they are used in the parameterization of frames to be sent, this will generate an error message.

➤ **?1, ?2,... , ?9:** the next 1, 2,..., 9 bytes will be ignored when analyzing the frame.

*Example:* it is parameterized the sending of a "1" value through the 1 bit communication objet number 2 if SKX receives the following frame through the serial port: "AA?223". The terminal equipment starts sending frames, including the following:

"AA857D23" → SKX sends the corresponding object with a "1" value, since the two bytes 857D are ignored, as it was specified by parameter, and the fixed part of the frame matches the parameterized one.

"AA112223" → same case as before. Now, SKX ignores the bytes 1122.


"AA685AB923" → SKX Advance will not send the value of the object, since the fixed part of the received frame does not match the parameterized one. The device will ignore the 2 bytes that follow AA (685A), as configured, and it will interpret B923 as the fixed part of the frame and, as it does not match the defined 23, the object number 2 will not send the "1" value.

➤ **\*\*:** it indicates that **zero or more bytes** of any value will appear until finding the fixed part of the frame defined by parameter (excluding the keys @h, @f, @1, @2). Therefore, the \*\* characters will represent the minimum set of characters found before the detection of the constant part after \*\*.

*Example:* it is parameterized the sending of the value "50" through the 1 byte communication object number 41 when receiving through the serial port the frame: "23\*\*AB". The terminal equipment sends this:

“231214AB” → SKX Advance starts analyzing the frame: 23 and AB are the fixed parts of the frame and all the bytes found before AB (constant part after \*\* in the frame definition) will be the characters defined as \*\*. The frame matches the frame configured by parameter, so the object number 41 sends the value 50.

“23ABAB” → in this case, SKX Advance interprets that \*\* in the empty set, as there are no characters between 23 and the first AB of the frame. SKX continues analyzing the frame and it finds the AB characters again, thus interpreting a mismatch between this frame and the parameterized one, so it does not send the value 50 through the object 41.

 **Special characters for transmission and reception:** these characters can be used in the definition of incoming or outgoing frames.

➤ **Subframes Keys:** they are defined this way “@ + character” and represent the inclusion of one of the frames defined in the “Frame configuration” window. The different possibilities are:

- @h: includes a header frame.
- @f: includes a footer frame.
- @1, @2: includes the subframe 1 or 2, respectively.
- @c: includes the checksum byte (or bytes).

➤ **##:** this character indicates the variable part associated to the communication object.

**1 bit objects:** it cannot be used.

**1 byte objects:** ## will always be 1 byte long. There are 4 cases in which this character can be used:

- Send variable frame: SKX Advance will automatically insert the value of the corresponding communication object in the position of the frame occupied by the character ##.

*Example:* it is defined by parameter this variable frame: “1234##55”. The 1 byte communication object number 44 receives the value 16 (decimal) through the KNX bus. Therefore, SKX Advance will send to the terminal equipment the frame “12341055”.

**(Note:** SKX converts the 16 decimal value into its hexadecimal equivalent (0x10) and inserts it in the

position occupied by ## in the ETS definition of the frame to sent it properly).

- Get variable object: the communication object will have the value received on the position occupied by the special character in the frame sent from the terminal equipment.

*Example:* it is defined the following frame associated to the 1 byte object number 48: "AA##E8". The terminal equipment sends this to SKX: "AA02E8", so the object will have the value 2 and will send it through the KNX bus.

- Send variable frame (%): similar to send a variable frame, but the value of the communication object will be preciously converted from KNX percentage (0-255) to the standard one (0-100 %).
- Get variable object (%): similar to get a variable object, but the variable byte will be converted from standard percentage (0-100) to KNX percentage (0-255).

**Note:** when sending and receiving fixed frames, the character ## cannot be used.

**14 bytes objects:** ## will always represent a chain of characters. In the case of frame transmission, this will be the point where SKX Advance should copy the characters received from the KNX bus; in the case of reception, this will be the starting point of the characters frame that SKX Advance must copy in the corresponding communication object.

In the sections 2.3.1, 2.3.2 y 2.3.3 there are some examples about the use of special characters with the SKX Advance objects.



## 2.3. PARAMETERS GROUPS

The communication object groups to use can be enabled here.

SKX Advance has a total of 65 communication objects, distributed this way:

| Size     | Number of groups | Number of objects per group |
|----------|------------------|-----------------------------|
| 1 bit    | 4                | 10                          |
| 1 byte   | 2                | 10                          |
| 14 bytes | 1                | 5                           |

There are 40 1-bit communication objects, whose object numbers are in the range 0 to 39; 20 1-byte objects, between 40 and 59, and 5 14-bytes objects, with numbers between 60 and 64.

When enabling the parameters groups, the parameter boxes corresponding to each of them are shown:

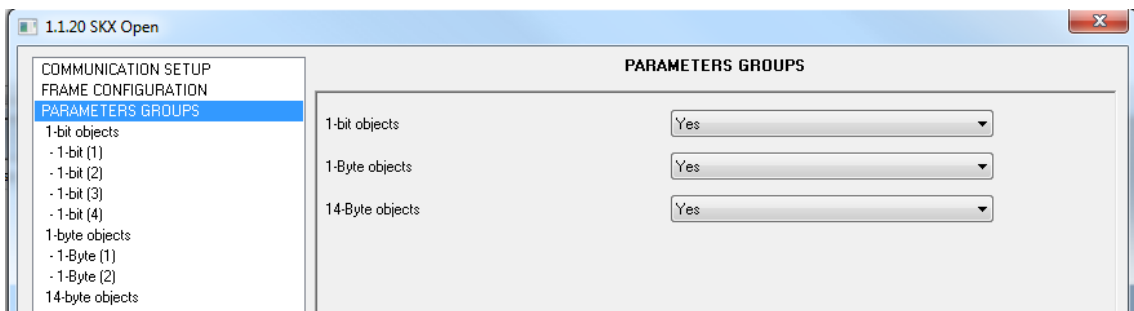


Figure 2.7. SKX Advance: Parameters groups

It is necessary to enable the objects with which work, inside every parameters group, and to configure the options for every kind of communication object.

Next, the different communication object types available are explained:

## 2.3.1. 1 BIT OBJECTS

The 1 bit objects allow sending a data frame from SKX to the terminal equipment, through RS-232, when SKX receives through the KNX bus a value previously parameterized in ETS (0 or 1) for the configured object (objects with number from 0 to 39). Besides, this kind of objects also allows the interface to send a value (0 or 1) through a specific communication object (0-39) when SKX receives a fixed frame from the terminal equipment, through RS-232.

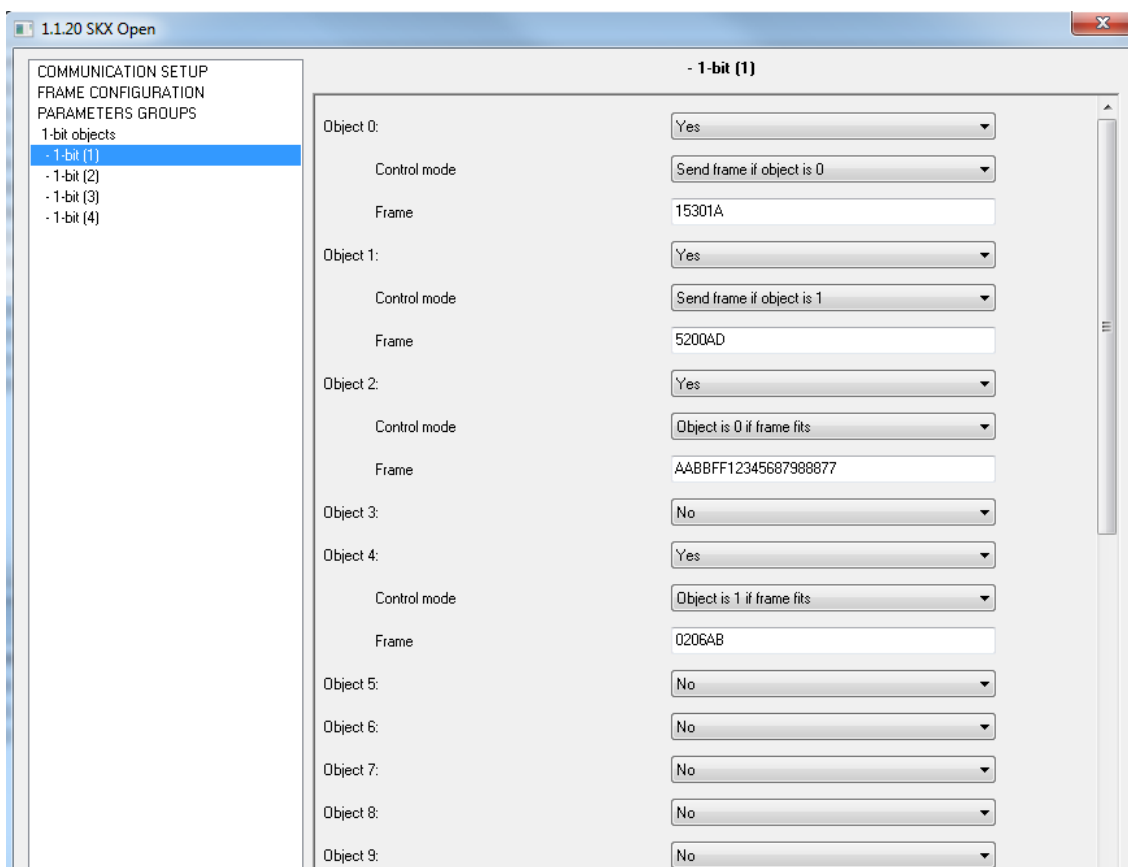


Figure 2.8. 1 bit communication objects. Group 1

For instance, with the previous configuration, SKX will send to the terminal equipment through RS-232 the frame “15301A” when SKX receives through the KNX bus the object number 0 with a 0 value. And if the object number 1 receives a 1, SKX will send the frame “5200AD” to the terminal equipment.

Likewise, if the interface receives through RS-232 the frame “AABBBFF12345687988877”, SKX will write a 0 in the object number 2, whereas if it receives the frame “0206AB”, SKX will write a 1 in the 1 bit object number 4.

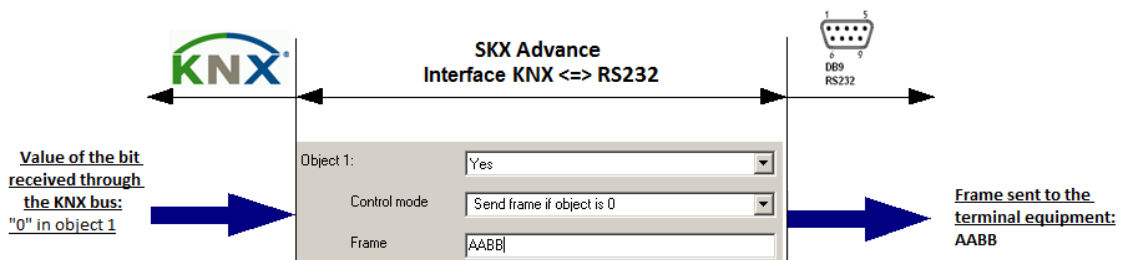
When enabling a 1 bit communication object, two options are shown: one for selecting the control mode and the other for defining the frame.

- **Object X. Control mode:** there are 4 control possibilities over every object, through the following parameters:

For the KNX → RS232 communication

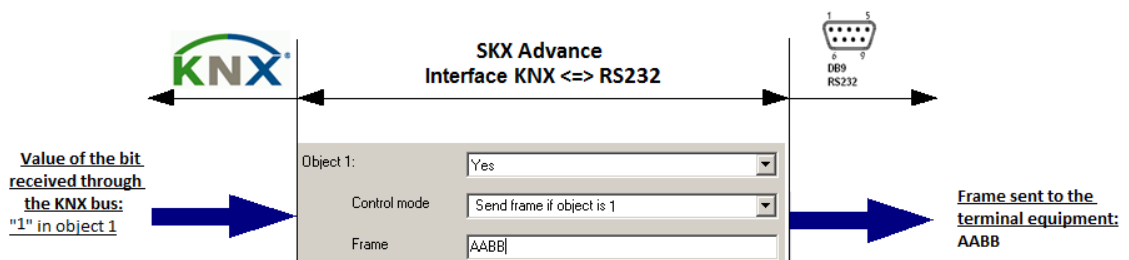
- **Send frame if object is 0:** Sending the frame (typed in the “Object X. Frame” parameter) to the terminal equipment when **receiving a “0”** through the KNX bus in the corresponding communication object.

*Example:*



- **Send frame if object is 1:** Sending the frame (typed in the “Object X. Frame” parameter) to the terminal equipment when **receiving a “1”** through the KNX bus in the corresponding communication object.

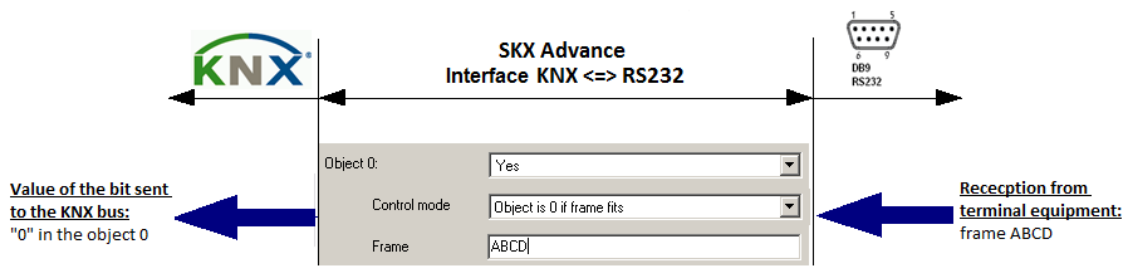
*Example:*



For the RS232 → KNX Communication

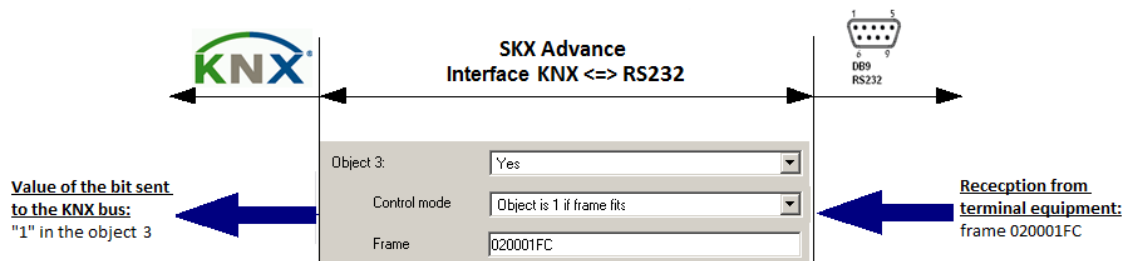
- **Object 0 if frame fits:** Sending a “0” through the object when receiving through the serial port a frame that fits the parameterized one.

Example:



- **Object is 1 if frame fits:** Sending a “1” through the object when receiving through the serial port a frame that fits the parameterized one.

Example:



- **Object X. Frame:** to define the frames for the communication. They should comply with the requirements mentioned in the section “2.2.1. Frame definition”.

### 2.3.2. 1 BYTE OBJECTS

The 1 bit objects allow sending a data frame from SKX to the terminal equipment, through RS-232, when SKX receives through the KNX bus a value previously parameterized in ETS (0-255) for the configured object (objects with number from 40 to 59). Besides, this kind of objects also allows the interface to send a value (0-255) through a specific communication object (40-59) when SKX receives a specific frame from the terminal equipment, through RS-232. These frames can be fixed or variable, depending on the value of the object and the same for the object value, which can be fixed or dependent on the received frame.

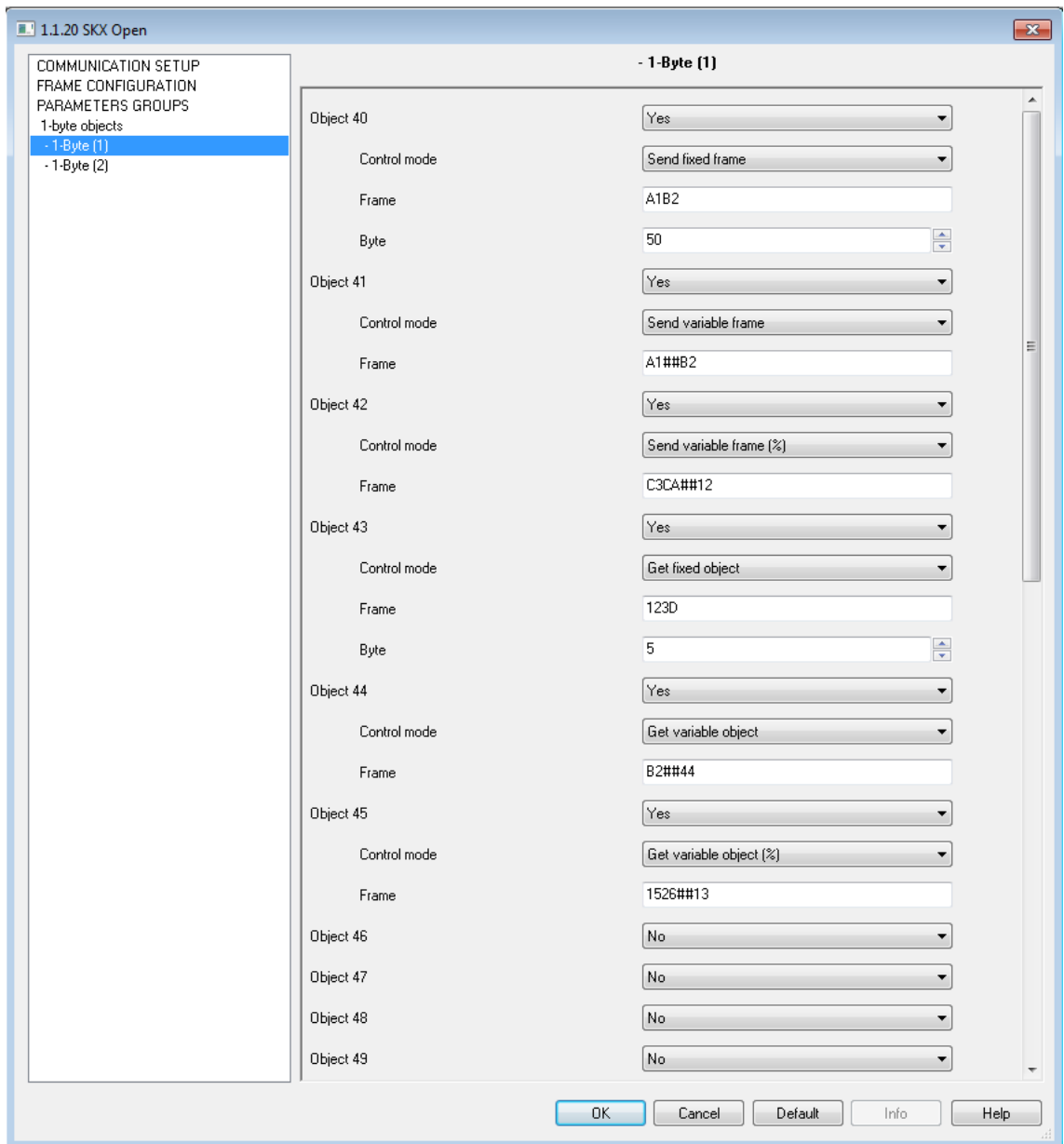


Figure 2.9. 1 byte communication objects. Group 1

For instance, with the previous configuration, SKX will send to the terminal equipment the frame “A1B2” through RS-232, when receiving through the KNX bus the object number 40 with the value 50 (decimal). Likewise, when SKX receives the frame “123D” through RS-232 from the terminal equipment, it will write the value 5 in the 1 byte object number 43.

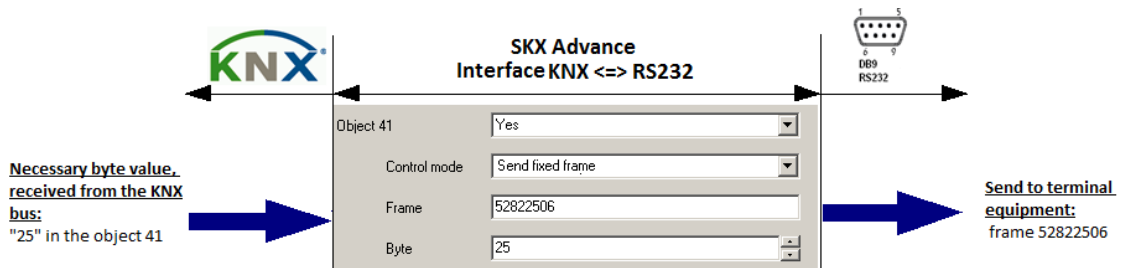
When enabling a 1 byte communication object, two options are shown: one for selecting the control mode and the other for defining the frame

- **Object X. Control mode:** there are six control possibilities over every object, through the following parameters:

For the KNX → RS232 Communication

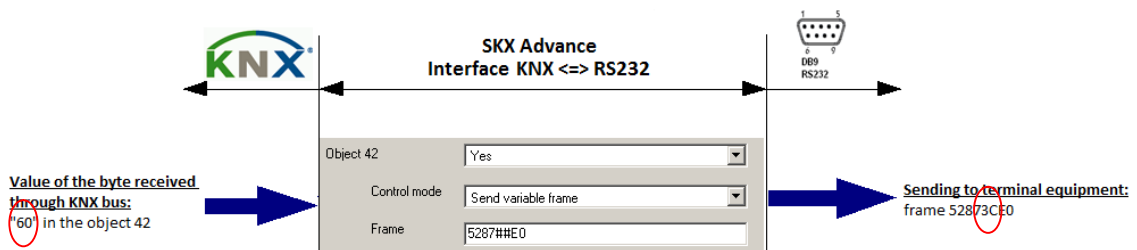
- **Send fixed frame:** Sending the frame (introduced in the “Object X. Frame” parameter) to the terminal equipment when receiving the value parameterized in the object. When enabling this kind of control, a new option appears: “**Object X. Byte**”, where it is defined the byte SKX expects to receive through the communication object to send the defined frame.

Example:



- **Send variable frame:** Sending the frame (introduced in the “Object X. Frame” parameter) to the terminal equipment when receiving a value through the communication object (via the key ##). The sent frame will vary depending on the value received through that object and it will be the same as the parameterized one, replacing ## by the value of the byte received through the corresponding communication object.

Example:

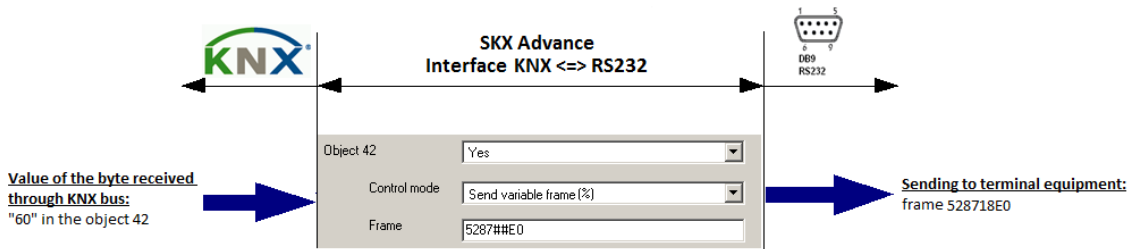


**Note:** here, SKX Advance receives the value “60” through the KNX bus. This is a **decimal** value. However, in the frame sent to the terminal equipment it appears the value “3C” at the position occupied by ## in the frame definition. “3C” is the **hexadecimal** value of decimal “60”. SKX Advance carries out this conversion to correct sending the frame.

- **Send variable frame (%):** sending the frame (the one typend on the parameter “Object X. Frame”) to the terminal equipment when receiving a value through the corresponding communication object (via the key ##). The received value will be converted into a number between 0 and 100 (%) and it will be incorporated to the frame

defined by parameter. The sent frame will vary, depending on the value received through the object (in percentage) and it will be the same as the one defined by parameter, replacing ## by the received value through the corresponding communication object (in percentage).

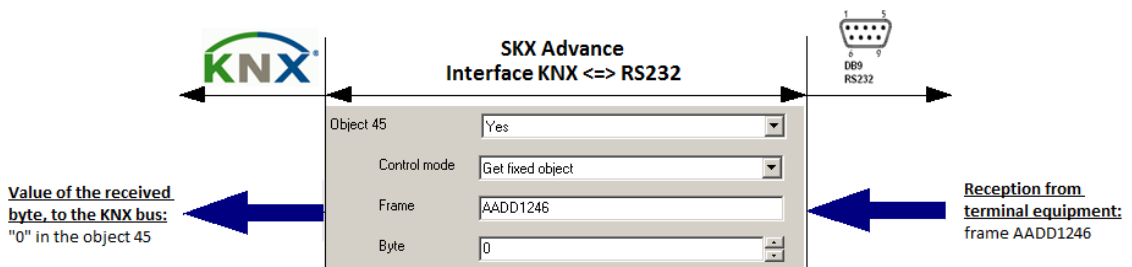
*Example:*



**Note:** in this case, SKX receives the decimal value “60”. This value in a KNX percentage (0-255), which is converted to standard percentage (0-100%) through a simple rule of three ( $60 \cdot 100 / 255$ ). Its equivalent value is 24%. This value is converted to an hexadecimal one: “18”. This value will replace the characters ## in the frame defined by parameter.

For the RS232 → KNX Communication

- **Get fixed object:** sending the parameterized value through the

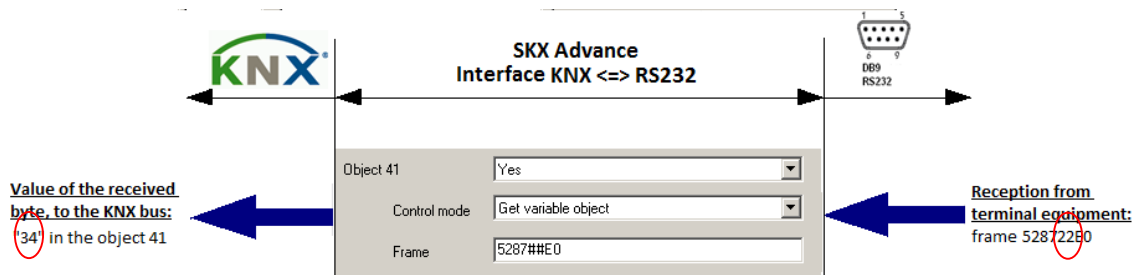


corresponding object when receiving through the serial port a frame that fits the one introduced by parameter. When enabling this kind of control, a new option is shown: “Object X. Byte”, where it is defined the byte SKX will write in the communication object when it receives the frame.

*Example:*

- **Get variable object:** sending a value through the corresponding communication object when receiving through the serial port a frame that fits the one introduced by parameter. The value of the communication object will be the corresponding to the part ## received in the frame.

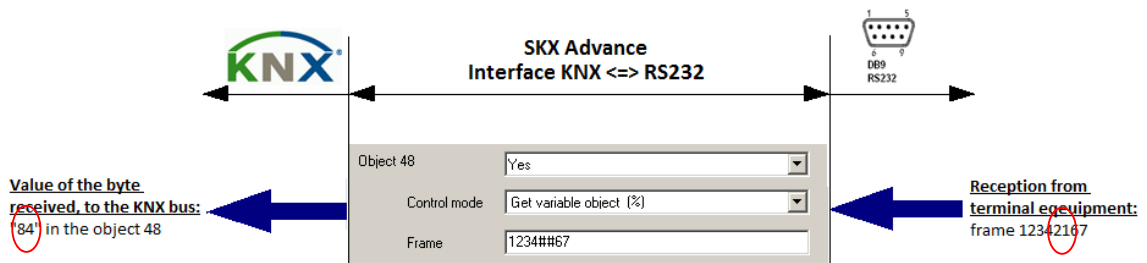
Example:



**Note:** the terminal equipment sends the frame 528722E0 in hexadecimal. The value that will take the communication object number 41 will be the corresponding to the ## received part, in this case, "22" (in hexadecimal), but converted to its decimal value, "34".

- **Get variable object (%):** sending a value to the object when receiving through the serial port a frame that fits the one introduced by parameter. The value of the communication object will be the corresponding to the ## received part of the frame (in percentage).

Example:



**Note:** SKX receives the hexadecimal value "21" in the corresponding part of ## in the defined frame. This value corresponds to the decimal value "33", this time as standard percentage (0-100%), that will be converted to



*KNX percentage, by means of a simple rule of three ( $33 \cdot 255 / 100$ ). Its equivalent value is "84" and it will be sent to the corresponding communication object.*

- **Objecto X. Frame:** to define the frames for the communication. They should comply with the requirements mentioned in the section "2.2.1. Frame definition".

### 2.3.3. 14 BYTES OBJECTS

The 14 bytes objects allow sending or detecting text strings into the serial port frames.

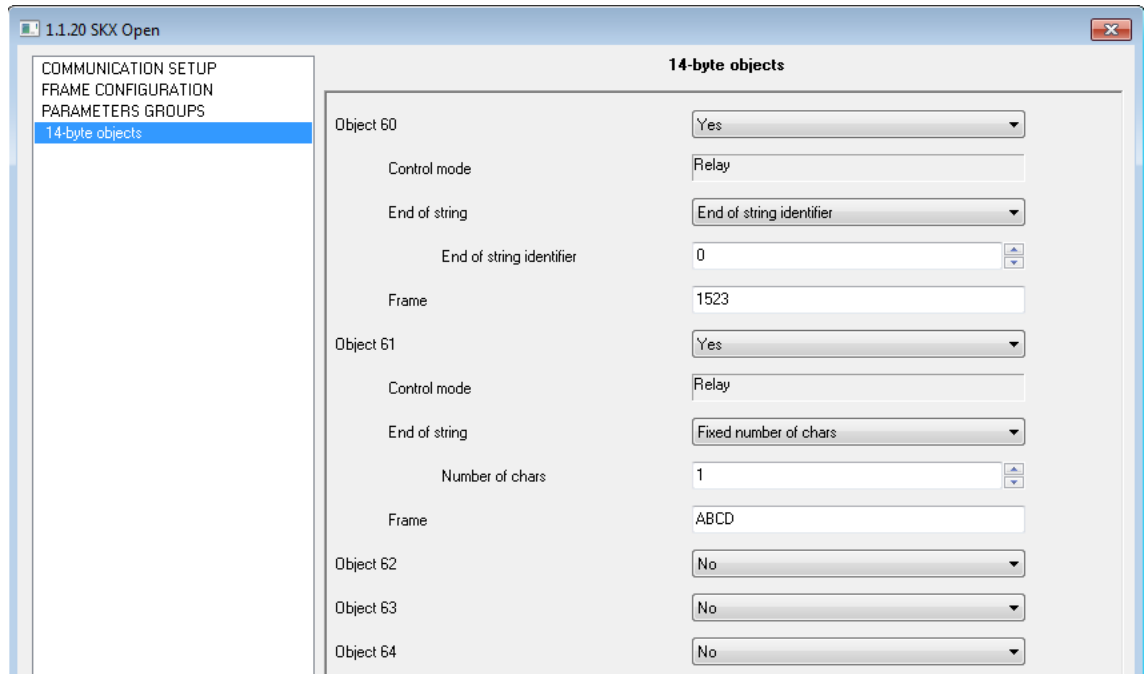
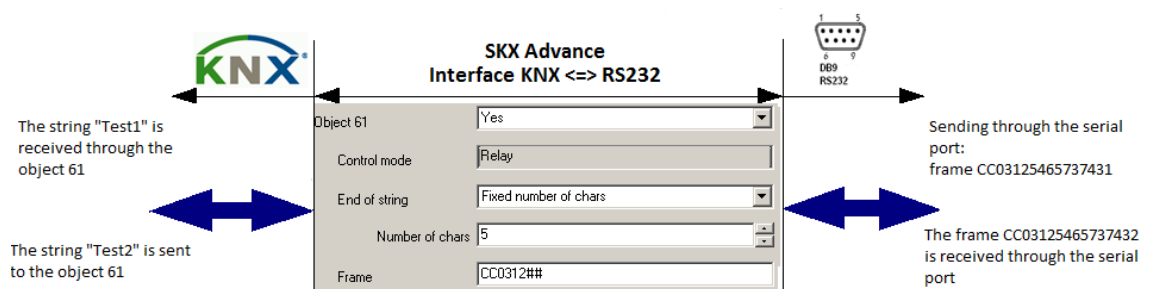


Figure 2.10. 14 bytes communication objects group

When enabling a 14 bytes object, three options are shown to select the control mode, the end of string type and to define the corresponding frame.

- **Object X. Control mode:** it exists only one way to control every object, through the **Relay** parameter, which allows transmitting again a variable text string received through the serial port through the KNX bus and vice versa (via the characters ##).
- **Object X. End of string:** to select the way to confirm the end of the string. There are two options:
  - **Number of chars:** it is defined the fixed number of characters (1 to 14) that the sent or received string must have.

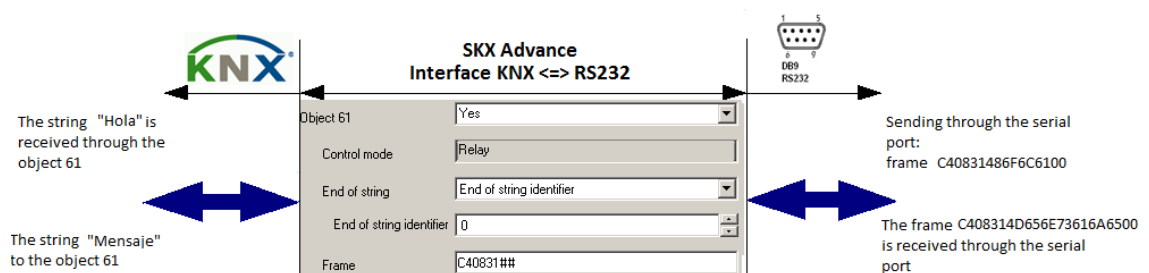
Example:



**Note:** in this example, the string "Test1", 5 characters, is received through the communication object number 61. The associated reaction is sending to the serial port the following frame: "CC0312**5465737431**", where CC0312 is the fixed part of the defined frame and "5465737431" is the ASCII encoding, character by character, of the string "Test1". After this, SKX receives through the serial port the frame "CC0312**5465737432**", what means the sending of the string "Test2", of 5 characters, to the object number 61.

- **End of string identifier:** it is defined here the byte that indicates the end of the string. This byte is usually 0x00.

Example:



**Note:** the End of string identifier 0x00 has been defined (writing a 0 in the corresponding box). The object 61 receives the text string "Hola", which will provoke a sending through the serial port of the frame "C40831**486F6C6100**", where C40831 is the fixed part of the frame defined by parameter, 486F6C61 is the ASCII encoding, character by character, of the "Hola" string and 00 is the defined end of string identifier. After this, SKX receives through the serial port the frame "C40831**4D656E73616A6500**", which generates the sending of the "Mensaje" string to the object number 61.

- **Object X. Frame:** to define the frames for the communication. They should comply with the requirements mentioned in the section "2.2.1. Frame definition".

## 2.4. ERROR OBJECTS

SKX Advance has several 1 bit communication objects that inform about errors in the operation of the program, mainly due to poor parameterization, although these errors can also be produced during the communication. Whenever SKX Advance detects an error, it will send the corresponding 1 bit object and the 1 bit generic error object (“Error code”).

Below the different errors that may occur are described:

- **Error: odd length.** The set of characters of any frame introduced by parameter is odd. (**Note:** *remember that this does not mean that the number of bytes of every frame – frame length – must be even, but the total number of characters does, since there are two characters for every byte*).
- **Error: bad usage of ‘\*’ or ‘?’.** There is nothing constant after \*\* or the character after ‘?’ is incorrect.
- **Error: bad usage of ‘@’.** Error using the special character @.
- **Error: checksum.** There is nothing to calculate the checksum, for instance, because the frame has been configured just as @c, or because the checksum offset is too big.
- **Error: bad usage of ‘#’.** Error using ##. Possible sintaxis error or it is not possible to associate ## with variable data.
- **Error: not hexadecimal.** SKX found in a frame defined by parameter a character with a value different from 0-9 or A-F (for example, a lowercase).

**Note:** *all the previous errors are parameterization errors.*

- **Error: too long.** The length of the frame to be sent or the received frame is greater than the maximum allowable length: 29 bytes. (This error can be a parameterization or a communication one).
- **Error: reception.** Undefined error of reception through the serial port. (Communication error).

Whenever the bus power returns, SKX analyzes all the frames introduced by parameter and, if it detects a failure in the frames, it sends the corresponding error objects.

There will be **only one report for every type of error**; i.e., if an error of the same type is detected in two or more frames (for instance, a bad usage of ‘\*’), this will be reported once.

If a frame has several errors, all of them will be reported, except if the introduced frame is odd, in which case only the error “Odd length” will be reported. For instance, the frame “AaB” has an odd length error and a not hexadecimal error; SKX Advance will only report the odd length error, obviating the other.

SKX Advance also notifies errors when it tries to send a bad parameterized fram to the serial port RS-232. In thsio case, it **will only report the first detected error in the frame**. For instance, suppose SKX sends the frame “AAaB?B”; this frame has two errors (there is a not hexadecimal character and a bad usage of ‘?’), but when it is starting to transmit, the sending will be stopped when the not hexadecimal error is detected, reporting only this error.

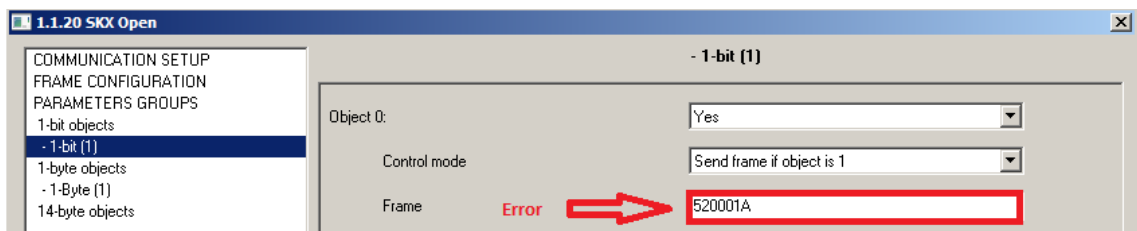
Therefore, SKX Advance notifies the detected errors in the frames after being programmed and every time wrong frames are received or are going to be transmitted.

## 2.4.1. ERROR EXAMPLES

Below a set of examples of the errors that SKX Advance is able to detect.

- **Odd length.** In the case of trying to send a frame with a set of characters of odd length, the 1 bit communication object “Error: odd length” will be enabled, with a “1” value. Besides, the 1 bit object “Error code” will be enabled and sent to the bus.

*Example:* it is defined in the object 0 the frame “520001A”, which length is odd (7 characters).



This will provoke the following error:

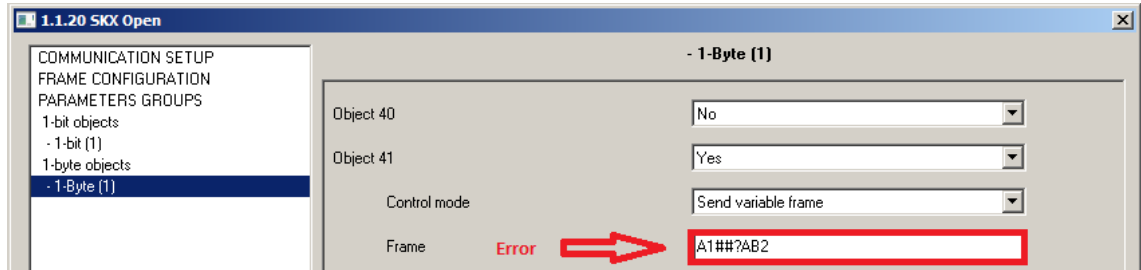
| # | Time         | Service  | F... | P.        | Src.addr  | Source | Dest.addr         | Destination | R | DPT   | Type  | Data |
|---|--------------|----------|------|-----------|-----------|--------|-------------------|-------------|---|-------|-------|------|
| 1 | 14:26:03.335 | to bus   | L    | 15.15.255 | Not Found | 1/1/0  | New Group Address |             | 6 | 1 bit | Write | \$01 |
| 2 | 14:26:03.355 | from bus | L    | 1.1.20    |           | 1/0/2  | ERROR CODE        |             | 6 | 1 bit | Write | \$01 |
| 3 | 14:26:03.375 | from bus | L    | 1.1.20    |           | 1/0/3  | ODD LENGTH        |             | 6 | 1 bit | Write | \$01 |

Figure 2.11. Error objects: Odd length

- **Bad usage of ‘\*’ or ‘?’.** If after the special character ‘?’ it is added a value other than [0-9], SKX Advance will activate the 1 bit communication object “Error: bad usage of ‘\*’ or ‘?’”, as well as the

“Error code” object. These objects will be also enabled if after the characters “\*\*” does not ppear anything constant.

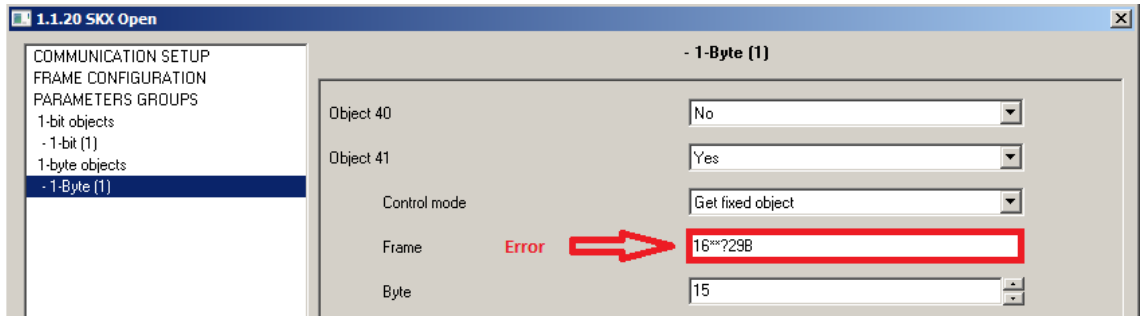
*Example I:* it is defined in the 1 byte object number 41 to send the frame “A1##?AB2”, where appears the character ‘A’ after ‘?’, which is an invalid value (since it must be a number between 0 and 9).



| # | Time         | Service  | F... | P. | Src_addr  | Source    | Dest_addr | Destination       | R | DPT    | Type  | Data       |
|---|--------------|----------|------|----|-----------|-----------|-----------|-------------------|---|--------|-------|------------|
| 1 | 14:42:17.249 | to bus   |      | L  | 15.15.255 | Not Found | 1/1/41    | New Group Address | 6 | 1 byte | Write | \$01   0 % |
| 2 | 14:42:17.269 | from bus |      | L  | 1.1.20    |           | 1/0/2     | ERROR CODE        | 6 | 1 bit  | Write | \$01       |
| 3 | 14:42:17.289 | from bus |      | L  | 1.1.20    |           | 1/0/4     | *?                | 6 | 1 bit  | Write | \$01       |

Figure 2.12. Error objects: bad usage of ‘?’

*Example II:* it is defined in the 1 byte object number 41 to write the value 15 when SKX receives from the terminal equipment the frame “16\*\*?29B”. SKX will provoke an error of bad usage of \*\*, since after \*\* there is nothing constant, but “?2”, which implies the appearance of two characters that can have any value.



|      |              |          |  |   |           |           |        |            |   |       |              |      |
|------|--------------|----------|--|---|-----------|-----------|--------|------------|---|-------|--------------|------|
| 1... | 14:51:45.190 | to bus   |  | S | 15.15.255 | Not Found | 1.1.20 |            | 6 | -     | T_Disconnect |      |
| 1... | 14:51:45.210 | from bus |  | L | 1.1.20    |           | 1/0/2  | ERROR CODE | 6 | 1 bit | Write        | \$01 |
| 1... | 14:51:45.230 | from bus |  | L | 1.1.20    |           | 1/0/4  | *?         | 6 | 1 bit | Write        | \$01 |

Figure 2.13. Error objects: bad usage of ‘\*\*’

**Bad usage of ‘@’.** The special character ‘@’ allows introducing headers, subframes or footers into a frame, and the keys to write all of them are already defined, and there cannot be other different from: @h, @f, @1, @2 and @c. in the case of introducing an invalid character

after @, SKX Advance will enable the 1 bit communication object “Error: bad usage of ‘@’”, as well as the object “Error Code”.

*Example:* it is defined the following frame to send if the object number 1 has the value “1”: “52@324”. The character 3 after @ is not valid, so SKX Advance will enable the corresponding error object and the frame will not be sent.

The screenshot shows the '1-bit (1)' configuration window. The 'Frame' field contains the text '52@324'. Below the window is a log table with the following data:

| # | Time         | Service  | F... | P. | Src.addr  | Source    | Dest.addr | Destination       | R | DPT   | Type  | Data |
|---|--------------|----------|------|----|-----------|-----------|-----------|-------------------|---|-------|-------|------|
| 1 | 07:22:02.087 | to bus   | L    |    | 15.15.255 | Not Found | 1/1/1     | New Group Address | 6 | 1 bit | Write | \$01 |
| 2 | 07:22:02.107 | from bus | L    |    | 1.1.20    |           | 1/0/2     | ERROR CODE        | 6 | 1 bit | Write | \$01 |
| 3 | 07:22:02.127 | from bus | L    |    | 1.1.20    |           | 1/0/5     | @                 | 6 | 1 bit | Write | \$01 |

Figure 2.14. Error objects: bad usage of ‘@’

**Checksum.** This error reports that it is not possible to perform the checksum (if it was configured to perform it, in the Frame Configuration) of a frame because there are no data (the defined frame is empty) or because the configured offset is too big. If any of these situations occurs, SKX Advance will enable the 1 bit communication object “Error: Checksum”, as well as the object “Error code”.

The screenshot shows the '1-bit (1)' configuration window. The 'Frame' field is empty, and a red arrow points to it with the word 'Error' next to it. Below the window is a log table with the following data:

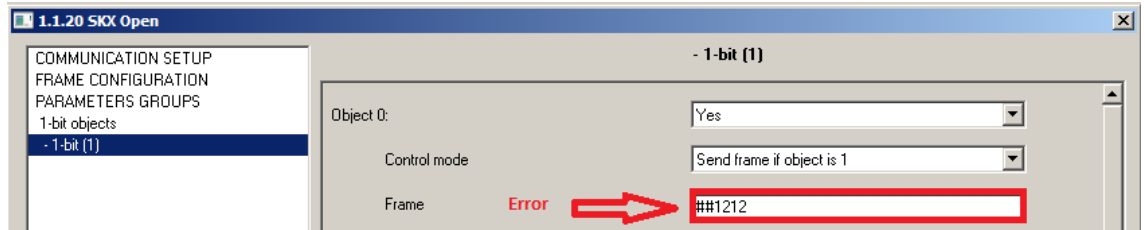
| # | Time         | Service  | F... | P. | Src.addr  | Source    | Dest.addr | Destination              | R | DPT   | Type  | Data |
|---|--------------|----------|------|----|-----------|-----------|-----------|--------------------------|---|-------|-------|------|
| 1 | 07:46:17.656 | to bus   | L    |    | 15.15.255 | Not Found | 2/0/0     | Nueva Dirección de Grupo | 6 | 1 bit | Write | \$01 |
| 2 | 07:46:17.676 | from bus | L    |    | 1.1.20    |           | 1/0/2     | ERROR CODE               | 6 | 1 bit | Write | \$01 |
| 3 | 07:46:17.696 | from bus | L    |    | 1.1.20    |           | 1/0/6     | CHECKSUM                 | 6 | 1 bit | Write | \$01 |

Figure 2.15. Definition of an empty frame. It will generate a checksum error.

**Bad usage of ‘#’.** Whenever this character is used in a wrong way, SKX Advance will enable the 1 bit object “Error: bad usage of ‘#’” and the object “Error Code”.

*Example:* the characters ### are used in the definition of a frame in the 1 bit object number 0. SKX Advance enables the corresponding error

object, since it tries to add a variable part and this is not possible for this kind of objects.



| # | Time         | Service  | F... | P.        | Src.addr  | Source | Dest.addr | Destination              | R | DPT   | Type  | Data |
|---|--------------|----------|------|-----------|-----------|--------|-----------|--------------------------|---|-------|-------|------|
| 1 | 07:51:01.113 | to bus   | L    | 15.15.255 | Not Found | 2/0/0  |           | Nueva Dirección de Grupo | 6 | 1 bit | Write | \$01 |
| 2 | 07:51:01.133 | from bus | L    | 1.1.20    |           | 1/0/2  |           | ERROR CODE               | 6 | 1 bit | Write | \$01 |
| 3 | 07:51:01.153 | from bus | L    | 1.1.20    |           | 1/0/7  |           | #                        | 6 | 1 bit | Write | \$01 |

Figure 2.16. Frame definition with a bad usage of ##

- **Frame too long.** Since the headers, footers and subframes allow lengths of n bytes, it is possible to get a frame definition greater than 29 bytes. If this happens, SKX Advance will enable the 1 bit object “Error: too long” and the object “Error Code”.

*Example:* it is defined a header of 10 bytes, a footer of 10 bytes and a subframe of 4 bytes. When adding a frame of 10 bytes to the header, footer and subframe, it exceeds the permissible length limit for a frame, so SKX Advance will enable the corresponding communication object.

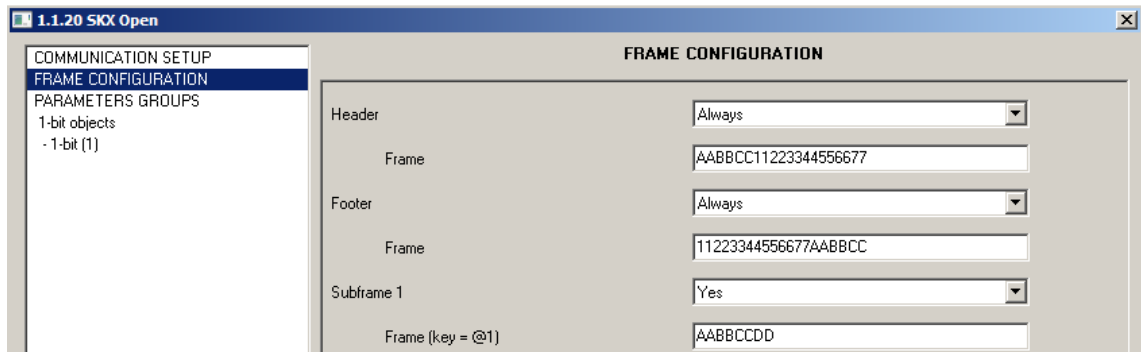
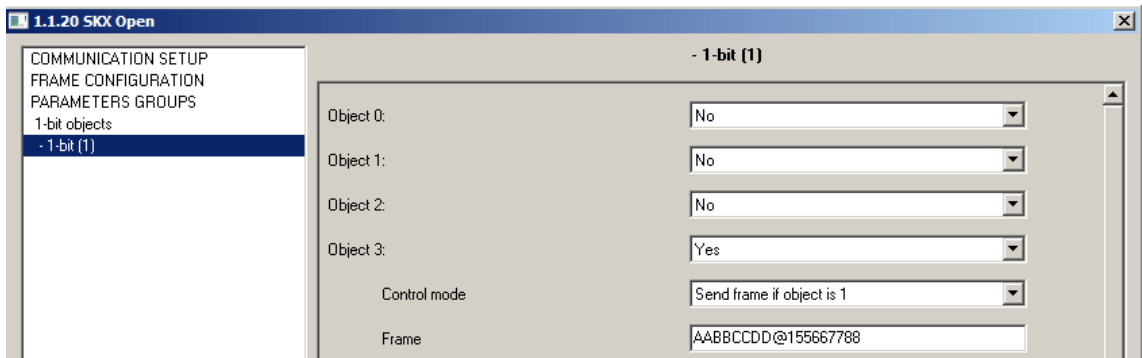


Figure 2.17. Definition of the header, footer and subframe 1





| # | Time         | Service  | F... | P. | Src.addr  | Source    | Dest.addr | Destination       | R | DPT   | Type  | Data |
|---|--------------|----------|------|----|-----------|-----------|-----------|-------------------|---|-------|-------|------|
| 1 | 08:02:58.473 | to bus   |      | L  | 15.15.255 | Not Found | 1/1/3     | New Group Address | 6 | 1 bit | Write | \$01 |
| 2 | 08:02:58.493 | from bus |      | L  | 1.1.20    |           | 1/0/2     | ERROR CODE        | 6 | 1 bit | Write | \$01 |
| 3 | 08:02:58.513 | from bus |      | L  | 1.1.20    |           | 1/0/8     | TOO LONG          | 6 | 1 bit | Write | \$01 |

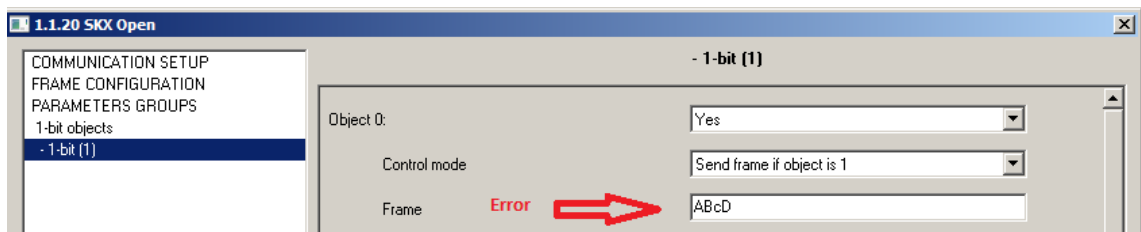
Figure 2.18. Definition of a frame too long, which will provoke the corresponding error

- Reception error.** When the parameters configuration of the serial communication does not match with the configuration of the received frames (different velocity, parity, etc.) SKX Advance will enable the 1 bit communication object “Error: reception” and the object “Error code”.

*Example:* the terminal equipment has been configured with a velocity of 9600 bauds, whereas the SKX Advance velocity is 1200 bauds. This will provoke that SKX Advance enables the reception error.

- Not hexadecimal character.** If a not hexadecimal character has been introduced when defining a communication frame (like a lowercase letter), SKX Advance will enable the communication objects “Error: not hexadecimal” and “Error code”.

*Example:* one the characters used to define the frame of the communication object number 0 is a lowercase letter: “ABcD”. This will make SKX Advance torable the corresponding error object.



| # | Time         | Service  | F... | P. | Src.addr  | Source    | Dest.addr | Destination              | R | DPT   | Type  | Data |
|---|--------------|----------|------|----|-----------|-----------|-----------|--------------------------|---|-------|-------|------|
| 1 | 08:13:36.154 | to bus   |      | L  | 15.15.255 | Not Found | 2/0/0     | Nueva Dirección de Grupo | 6 | 1 bit | Write | \$01 |
| 2 | 08:13:36.174 | from bus |      | L  | 1.1.20    |           | 1/0/2     | ERROR CODE               | 6 | 1 bit | Write | \$01 |
| 3 | 08:13:36.194 | from bus |      | L  | 1.1.20    |           | 1/0/10    | NOT HEXADECIMAL          | 6 | 1 bit | Write | \$01 |

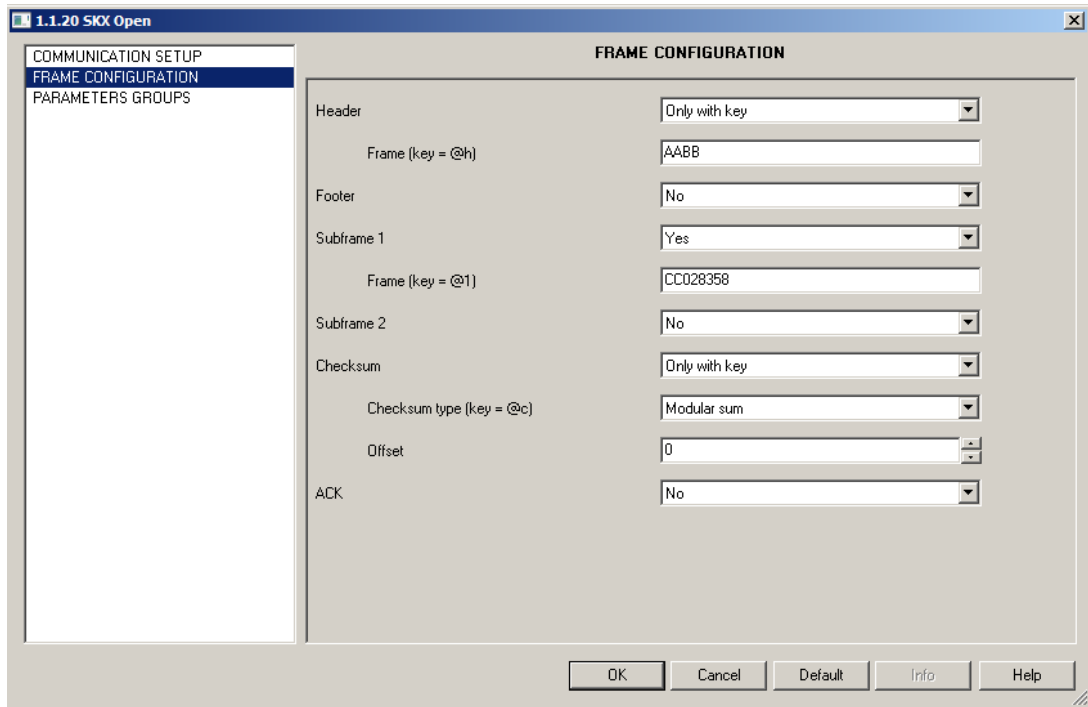
Figure 2.19. Frame definition with a not hexadecimal character

## 2.5. CONFIGURATION EXAMPLES

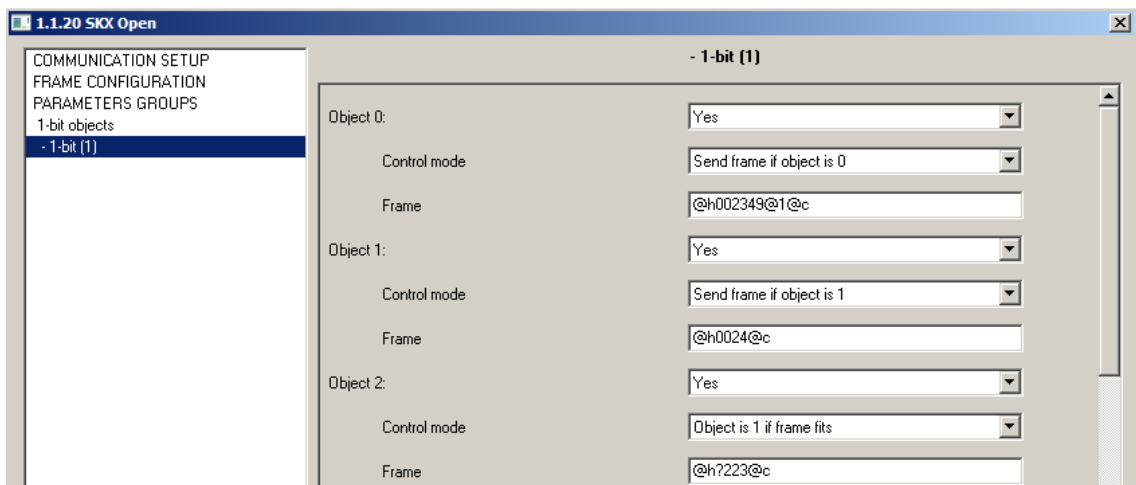
As a summary, there are below some examples of frames configuration in SKX Advance and examples of use, to better understand its operation.

### 2.5.1. 1 BIT OBJECTS

In the “Frame Configuration” window, the following special frames are defined:



After this parameterization, the 1 bit objects to be used are defined. In this case, the objects number 0 and the number 1 are enabled to send frames to the terminal equipment through RS-232 and the object number 2, to receive frames from the terminal equipment. The parameterization of these objects is as follows:



In the following table it is shown a possible operating sequence, with the sent and received frames in each case. (**Note:** the bold characters indicate the Checksum calculated as modular sum of the frame characters).

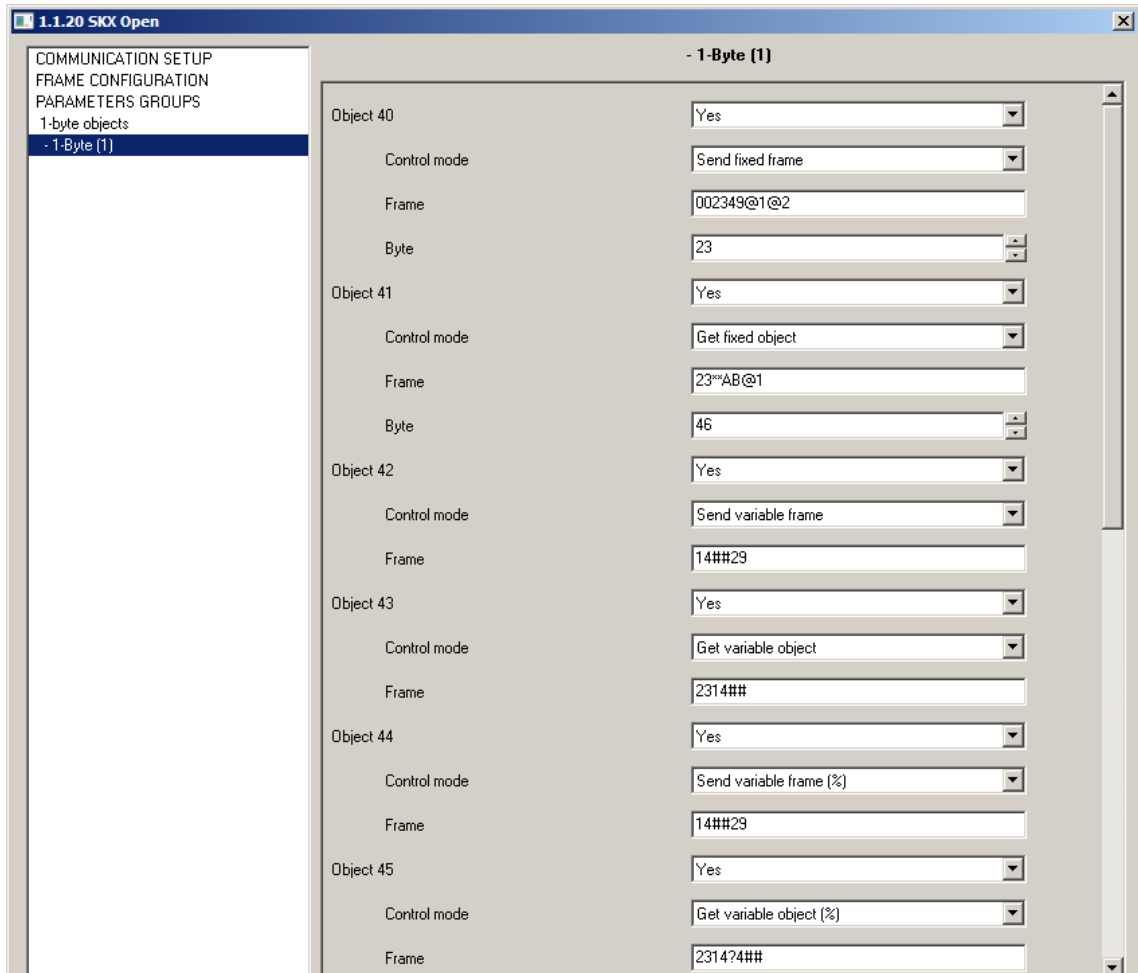
| EVENT   | REACTION  |
|---|---|
| The object 0 receives a "1" value through the KNX bus   | No action, since the object number 0 only sends the parameterized frame when it receives a "0"                            |
| The object 0 receives a "0" value through the KNX bus   | SKX sends through the serial port RS-232 to the terminal equipment the frame "AABB002349CC028358 <b>7A</b> " <sup>*</sup> |
| The object 1 receives a "1" value through the KNX bus   | SKX sends through the serial port RS-232 to the terminal equipment the frame "AABB0024 <b>89</b> "                        |
| SKX receives from the terminal equipment through the serial port the frame: "0024"                  | No action. The frame does not fit the parameterized one   |
| SKX receives from the terminal equipment through the serial port the frame: "AABB383223 <b>49</b> " | No action, since the received checksum is not correct (it should be <b>F2</b> )   |
| SKX receives from the terminal equipment through the serial port the frame: "AABB383223 <b>F2</b> " | SKX writes a "1" in the object number 2   |
| SKX receives from the terminal equipment through the serial port the frame: "AABB856D23 <b>7A</b> " | SKX writes a "1" in the object number 2   |

<sup>\*</sup> Checksum = **7A** = AA+BB+00+23+49+CC+02+83+58

## 2.5.2. 1 BYTE OBJECTS

In the "Frame Configuration" window, the following special frames are defined:

After this parameterization, the 1 byte objects to be used are defined. In this case, the objects number 40, 41, 42, 43, 44, 45 and 46 are enabled, each of them with a different task. The parameterization of these objects is the following:



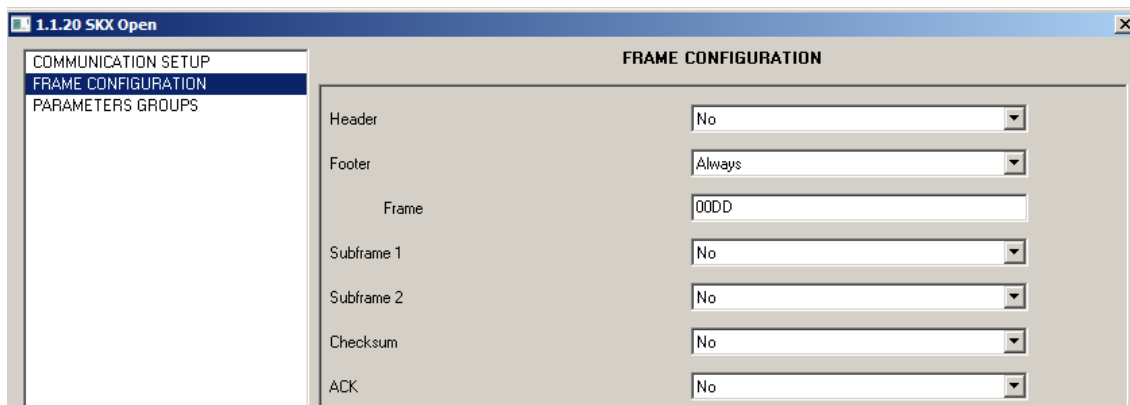
In the table below it is shown a possible operating sequence, with the sent and received frames in each case.

| EVENT  | REACTION   |
|--|--|
| The object 0 receives a "12" through the KNX bus   | No action, since the object 40 only sends the parameterized frame if it receives a "23"  |
| The object 0 receives a "23" through the KNX bus   | SKX sends through the serial port RS-232 to the terminal equipment the frame "AABB002349CC028358DD00"  |
| SKX receives from the terminal equipment through the serial port the frame: "3232"                 | No value is sent to the object, since the frame does not fit any of the parameterized.<br>SKX sends the ACK frame = "0033"   |
| SKX receives from the terminal equipment through the serial port the frame: "2348"                 | No value is sent to the object, since the frame does not fit any of the parameterized.<br>The ACK frame will not be sent, since the frame "2348" fits the parameterized ACK reply  |
| SKX receives from the terminal equipment through the serial port the frame: "23ABCC028358"         | No value is sent to the object 41, although the received frame fits the defined in the object. But the header (AABB) has not been sent, and it is compulsory to receive the frame OK.<br>SKX sends the ACK frame= "0033"   |
| SKX receives from the terminal equipment through the serial port the frame: "AABB23ABCC028358"     | SKX writes a "46" in the object number 41, since the received frame fits the parameterized one (in this case, ** is an empty set).<br>SKX sends the ACK frame= "0033"  |
| SKX receives from the terminal equipment through the serial port the frame: "AABB233333ABCC028358" | SKX writes a "46" in the object number 41, since the received frame fits the parameterized one (in this case, ** = "3333").<br>SKX sends the ACK frame= "0033"   |
| SKX receives from the terminal equipment through the serial port the frame: "AABB23ABABCC028358"   | SKX does not send any value to the object, because in this case **='AB', which fits the fixed part of the frame after **, so SKX Advance interprets it as an empty set and continues analyzing the frame, that will not fit the parameterized one, so SKX does not send anything except the ACK. |
| The object 42 receives through the KNX bus the value <u>16</u> (decimal)                           | SKX sends to the terminal equipment through RS-232 the frame: "AABB14 <u>10</u> 29" (the value <u>16</u> has been converted to its hexadecimal equivalent: 0x <u>10</u> )  |
| SKX receives from the terminal equipment through the serial port the frame: "AABB2314"             | No value is sent to the object, although the received frame fits the defined in the object 43. But there is no value in the ## part of the frame to indicate the variable part to write in the object.<br>SKX sends the ACK frame.   |
| SKX receives from the terminal equipment through the serial port the frame: "AABB2314 <u>93</u> "  | SKX writes in the object number 43 the decimal value (the equivalent of the hexadecimal value received in the ## part of the frame: 0x <u>93</u> → <u>147</u> ). SKX writes 147 in the object 43 and sends the ACK.  |

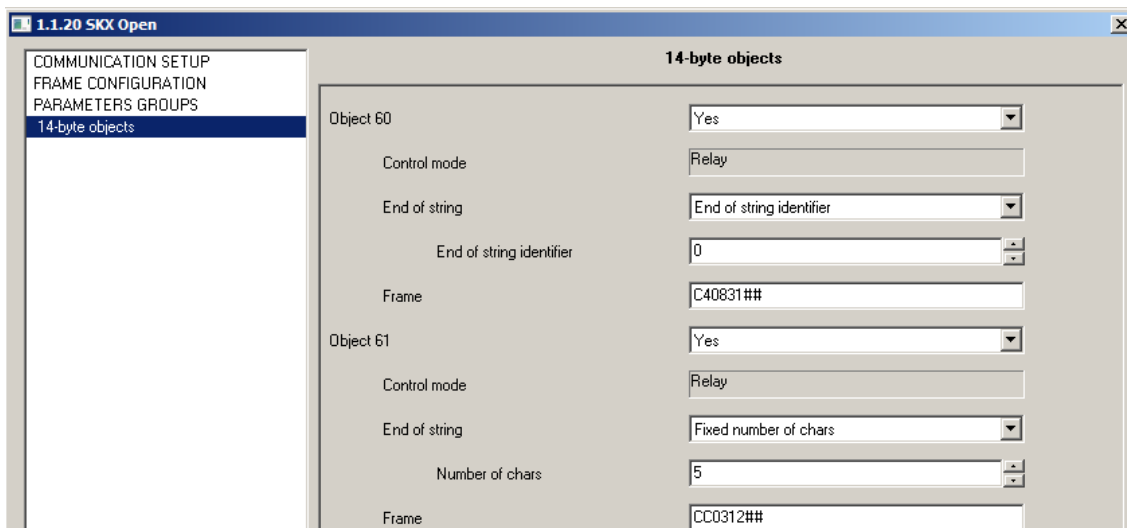
| EVENT   | REACTION   |
|---|--|
| SKX receives from the terminal equipment through the serial port the frame: "AABB2314 <u>28</u> "         | SKX writes in the object 43 the decimal value equivalent to the hexadecimal received in the ## part of the frame (0x <u>28</u> → <u>40</u> ).<br>SKX writes 40 in the object 43 and sends the ACK  |
| The object 44 receives through the KNX bus the value <u>83</u>  | SKX sends through RS-232 to the terminal equipment the frame "AABB14 <u>21</u> 29" (the received value <u>83</u> is in decimal and references a KNX percentage. SKX first converts it to standard percentage → $83 \cdot 100 / 255 \approx 33\%$ ; and then, to hexadecimal: 33d → 0x <u>21</u> )                    |
| The object 44 receives through the KNX bus the value <u>255</u>   | SKX sends through RS-232 to the terminal equipment the frame: "AABB14 <u>64</u> 29" (the received value <u>255</u> is in decimal and references a KNX percentage. SKX first converts it to standard percentage → $255 \cdot 100 / 255 = 100\%$ ; and then to hexadecimal: 100d → 0x <u>64</u> )                      |
| SKX receives from the terminal equipment through the serial port the frame: "AABB2314AABBCCDD <u>83</u> " | SKX writes in the object 45 the decimal value equivalent to the value received in the part ## of the frame → <u>83</u> . This is a standard percentage. SKX transforms it to KNX percentage ( $83 \cdot 255 / 100 = 212$ ) and this will be the value that SKX writes in the object 45.<br>SKX sends the ACK frame.. |

### 2.5.3. 14 BYTES OBJECTS

In the "Frame Configuration" window, the following special frames are defined:



After this parameterization, the 14 bytes objects to be used are defined. In this case, the objects number 60 and 61 are enabled. The parameterization of these objects is the following:



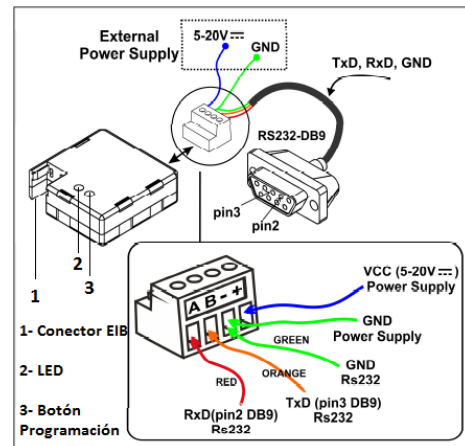
In the following table it is shown a possible operating sequence, with the sent and received frames in each case:

| EVENT   | REACTION   |
|---|--|
| The object 60 receives the string "Hola" through the KNX bus  | SKX sends to the terminal equipment through RS-232 the frame "C40831 <b>486F6C61</b> 0000DD", where:<br>Hola = " <b>486F6C61</b> " (ASCII encoding of each character)<br>00 = End of string identifier<br>00DD = Footer                |
| SKX receives from the terminal equipment through the serial port the frame: "C40831 <b>4D656E73616A6500</b> "     | No action is performed, since the received frame does not include the footer   |
| SKX receives from the terminal equipment through the serial port the frame: "C40831 <b>4D656E73616A650000DD</b> " | SKX writes the string "Mensaje" in the object number 60.   |
| The object 61 receives the string "Test" through the KNX bus  | SKX does not perform any action, since the string does not have the parameterized length, 5 characters   |
| The object 61 receives the string "Test1" through the KNX bus   | SKX sends to the terminal equipment through RS-232 the frame "CC0312 <b>5465737431</b> 00DD", where:<br>Test1 = " <b>5465737431</b> " (ASCII encoding of each character)<br>00DD = Footer  |
| SKX receives from the terminal equipment through the serial port the frame: "CC0312 <b>50727565626100DD</b> "     | SKX writes the string "Prueba" in the object 61  |
| SKX receives from the terminal equipment through the serial port the frame "CC0312 <b>5465737400DD</b> "          | No action is performed, since the frame has a fixed length (5 characters), and when SKX analyzes it, interprets "5465737400" as the incoming string; SKX searches next the footer, but it does not find it (since what remains is DD). |

### 3. PRODUCT SUMMARY



Device Hardware



SKX- RS232 Connection

#### SKX: ZENNIO interface.

- **Bidirectional** communication between the KNX bus and external devices with RS-232 (like televisions, surveillance systems, audio systems, etc.)
- Possibility of expanding an installation with devices that does not have KNX communication but RS-232.

#### SKX Advance: application program.

- Dynamic frames of variable length (up to **29 Bytes**)
- Communication objects of several types
- Communication errors detection
- Several transmission velocities
- **Adptable** to any protocol that governs the operation of the terminal equipment
- Great **versatility**.





**BECOME USER!**

<http://zennio.zendesk.com>

**TECHNICAL SUPPORT**