Technical specification

Xtender serial protocol

Date : 19.05.22

Version : V1.6.32 (R682)

403C\Technical specification - Xtender serial protocol.pdf

Contents

1. Introduction

This technical specification describes the protocol used to communicate with the Studer Innotec Xcom-232i communication module. It is also valid for the discontinuted RCC-02/-03 special execution ES N° 32 (RCC-02/-03-32).

1.1 Conventions used in this document

- Numbers that start with "0x" are in hexadecimal, like in the C integer litterals.
- Byte streams are represented with hexadecimal separeted by space like "00 01 1F 48"
- constant values are usually represented in UPPER CASE
- field names are in lower_case_with_underscore

1.2 List of acronyms

RCC	The Studer Innotec remote control used to configure the Xtender system
Xcom-232i	The Studer Innotec RS-232 communication module that has the function of a DCE, Data Communications Equipment
DTE	Data Terminal Equipment, the PC or controller system that wants to communicate with the Xcom-232i
SCOM	Naming prefix used for the Studer Innotec serial protocol

2. Physical layer

The physical layer is RS-232. The Xcom-232i is equiped with a DE-9 (also known as DB-9) Female connector which provides this interface.

The serial port is galvanically separated with an isolation of 500 V DC relative to the negative battery potential.

2.1 Connector Pinning

On the female connector of the Xcom-232i, only the wires “receive data”, “transmitted data” and ground are connected. The other wires are not connected, and the DTE must ignore signals such as CTS, DTR or DCD.

pin number	usage
1	not connected
2	RxD
3	TxD
4	not connected
5	GND
6	not connected
7	not connected
8	not connected
9	not connected

2.2 Cable to use

The cable to be used with a PC is a Female-Male, straight.

3. Data link layer

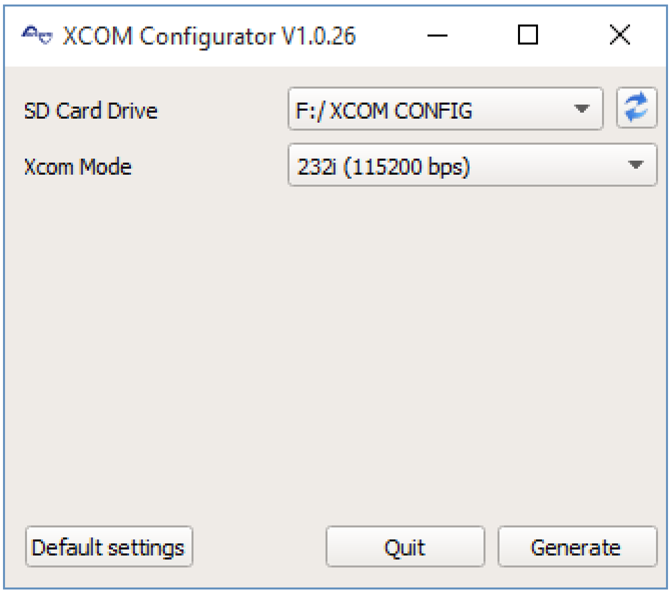
The data link layer, as defined in the OSI model, is used to send and receive frame on the RS-232.

3.1 RS-232 configuration

The default RS-232 configuration is defined as :

- A fixed baudrate of 38400 bps
- 1 start bit
- 8 bit of data, LSB first
- 1 parity bit
- even parity
- 1 stop bit

If you require a higher speed, it is possible to increase the Xcom-232i baudrate to 115200 bps. To do that, you require the last version of Xcom Configurator (available to download on the Studer official website under Download center->Software and Updates) in order to create the new configuration file. After the configuration file generation, insert the SD card in the powered Xcom-232i to upgrade his baudrate.



SD Card Drive: F:/ XCOM CONFIG

Xcom Mode: 232i (115200 bps)

Figure 1: Xcom Configurator

As the baudrate is 3 times bigger, the cable length is 3 times shorter. The maximal cable length is about 1.3 meters.

The Xcom-232i MUST have at least the software version 1.5.88 or 1.6.20 in order to support the configuration file.

3.2 Byte Endianness

All values are in little endian, i.e. LSB bytes are sent on the Physical layer first.

3.3 Frame

The Xcom-232i and the DTE exchange frames consist of a header of 14 bytes followed by a variable number of data bytes and 2 bytes of checksum.

start_-byte	**frame_-flags**	**src_-addr**	**dst_-addr**	**data_-length**	**header_-checksum**	**frame_-data**	**data_-checksum**
1 byte	1 byte	4 bytes	4 bytes	2 bytes = N	2 bytes	N bytes	2 bytes

start_byte

The start byte is always 0xAA

frame_flags:

BIT7-BIT6 : reserved.

BIT5 : is_datalog_supported, 1 if the datalogger is supported.

BIT4 : is_new_datalogger_file_present, 1 if there is a new datalog file on the SD card. This bit is reseted by the datalog read command through the SCOM or by extracting the SD card.

BIT3 : is_sd_card_full, 1 if the SD card is full. This bit is reseted by extracting the SD card.

BIT2 : is_sd_card_present, 1 if the SD card is present

BIT1 : 1 at each start or restart of the Xcom-232i, also after a WD Reset. This bit can be cleared with the RCC/Xcom-232i signal parameter {5104}.

BIT0 : is_message_pending flag, 1 if there are some messages pendings.

src_addr

src_addr is the source address, 32 bit little endian

dst_addr

dest_addr is the destination address, 32 bit little endian

data_length

the length of the frame's data, in byte

The maximum number of frame_data is 240 (so that 14+240+2 = 256)

header_checksum

the checksum of the header, from frame_flags to data_length (included)

frame_data

the data bytes

data_checksum

the checksum of all the data bytes of frame_data

3.4 Checksum algorithm

The checksum is computed with the following algorithm:

```
A = 0xFF
B = 0
For I FROM 0 TO number_of_bytes -1 DO
   A := (A + DATA[I]) mod 0x100;
   B := (B + A) mod 0x100;
END
checksum[0] := A
checksum[1] := B
```

A and B are byte values and the addition is made modulo 256.

After an invalid parity bit, header or data checksum, the data link layer is reseted and waits for an other frame.

3.5 Addressing the devices

Address	Devices	Remarks
0	Broadcast	
100	a virtual address to access all XTH, XTM and XTS	see section "multicast addresses"
101 to 109	a single XTH, XTM or XTS inverter	ordered by the index displayed on the RCC
191 to 193	virtual address to access properties on all inverters on a phase : 191 for L1, 192 for L2 and 193 for L3	a read access return the value of the master of the phase
300	a virtual address to access all VarioTrack	see section "multicast addresses"
301 to 315	VarioTrack	ordered by the index displayed on the RCC
501	Xcom-232i	alias for the gateway that the DTE uses to communicate (the Xcom-232i to which you speak with RS-232)
600	a virtual address to access all BSP	see section "multicast addresses"
601	BSP	
700	a virtual address to access all VarioString	see section "multicast addresses"
701 to 715	VarioString	ordered by the index displayed on the RCC

3.6 Multicast addresses

A WRITE_PROPERTY to this kind of address will have the effect to change the property value on all devices of the same kind. READ_PROPERTY operations are not supported.

3.7 Response delay

The response delay of the Xcom-232i can be up to 2 seconds. This is a good value for a timeout in the DTE implementation.

The reponse delay depends on the bus load (number of devices, number of RCC or Xcom-232i, values displayed on the RCC). The use of the datalogger on the Xcom-232i or on other RCC can cause a periodic increase of the reponse delay every 60 seconds.

3.8 Hardware Watchdog

The remote control parameter {5103} " Activation of the watchdog hardware (deactivation restarts the Xcom-232i)" allows activation or deactivation of the hardware watchdog, initially disabled. In case the Xcom-232i is not working properly a Reset will be initiated automatically. The bit 1 of the frame_flags (see 3.3) reflects a start or restart of the Xcom-232i. This bit can be cleared with the RCC/Xcom-232i parameter {5104} "Clears the restart flag of Xcom-232i".

This function reset only the Xcom-232i. For the Xtender, see the parameters {1628} and {1629} in section "XTENDER Watchdog".

3.9 SCOM Watchdog

The RCC/Xcom-232i parameters {5095} "Enable SCOM watchdog" and {5096} "SCOM watchdog delay before reset of Xcom-232i" allow configuration of the SCOM watchdog, initially disabled. There is two kind of security activated by this parameter. First, when a request is received a timer is started. If the response is not send after {5096} seconds, for example if an inverter present does not respond, the Xcom-232i is reset. A second timer is also activated when the first request arrives. It is set to zero every time a request is received. When it reaches {5096} seconds the Xcom-232i is reset. Before both kind of reset a message will be sent. Like the hardware watchdog, the bit 1 of the frame_flags (see 3.3) reflects a start or restart of the Xcom-232i and it can be cleared with the RCC/Xcom-232i parameter {5104} "Clears the restart flag of Xcom-232i".

3.10 XTENDER Watchdog

The parameters {1628} and {1629} make possible to set a watchdog function inside the inverter. The watchdog is a monitoring software that restarts the Xtender in case the communication is lost.

This system is active when the parameter {1628} (Xtender watchdog enable) is enabled and the parameter {1550} (parameters saved in flash memory) is disabled or if you write parameter with the property *unsaved_value_qsp*. Each time the CAN receives a parameter the counter, whose duration is set in seconds by parameter {1629}, will be restarted. If no parameter is received during this period of time the Xtender will stop and a RESET will take place. The device restarts in the configuration determined by the parameter settings before {1550} was set to “no”.

In a multi-unit system, each Xtender will handle this function independently. Each device must therefore receive a parameter within the time-frame set by {1629}.

By default this level is deactivated and the time period is set to 60s, adjustable from 10s to 300s.

3.11 VarioTrack Watchdog

The parameters {10342} and {10343} make possible to set a watchdog function inside the VarioTrack. The watchdog is a monitoring software that restarts the VarioTrack in case the communication is lost.

3.12 VarioString Watchdog

The parameters {14218} and {14219} make possible to set a watchdog function inside the VarioString. The watchdog is a monitoring software that restarts the VarioString in case the communication is lost.

3.13 Device identification

The RCC/Xcom-232i parameter {5119} allows to identify a device with the signaling system by flashing all of its LEDs. The written value corresponds to the Xtender’s or VarioTrack’ SCOM addresses. Unicast and multicast addresses are supported. For the Xtender, the phase addresses (191, 192 and 193) enable the signalling of all Xtenders on a given phase at the same time. The value 0 disables all signals. Sending a new value disables the previous value. If there is no more writing, all signals turn off after 45 seconds.

4. Application layer

The OSI layers 3 to 6 are not used. The application layer defines a number of « services ». A DTE sends a request frame and waits for a response frame from the Xcom-232i. If an error in the header checksum or data checksum is detected, there is no response from the application layer and the Xcom-232i waits for another request as if nothing has been received.

The Xcom-232i copies the src_addr of the request in the response dst_addr.

4.1 Services

The first two bytes of frame_data define the type of service and different flags for this service.

service_flags	**service_id**	**service_data**
1 byte	1 byte	N bytes

service_flags:

BIT7-BIT4 : reserved.

BIT1 : is_response flag, 0 if it is a request from the DTE to the Xcom-232i, 1 if it is response from the Xcom-232i.

BIT0 : error flag, 0 in case of success, 1 if an error occurred. In case of a request, error is always 0.

service_id:

One of the following services, described later in this document:

READ_PROPERTY = 0x01

service_data:

The data specific to the service. In case of a problem the errors are reported in a service-specific way, but the response has to include the error code described in the next section.

4.2 Object model

The different data accessible on each device are organized in object classes. Every object class has a number of properties. The service READ_PROPERTY is used to read the object’s properties.

4.2.1 READ_PROPERTY service

This service is used to read an object's property.

The DTE sends a request frame with the following frame_data:

service_flags	service_id	object_type	object_id	property_id	property_data
1 byte 0x00	1 byte 0x01	2 bytes	4 bytes	2 bytes	0 byte

service_flags : is_response =0, error=0

service_id : 0x01 for READ_PROPERTY

object_type : the object type identifier, defined later in this document

object_id : the object identifier, specific to each object type, i.e. two objects with different type can have the same id

property_id : identify the property in the object

property_data : no property data

The Xcom-232i responds with a frame with the following frame_data:

service_flags	service_id	object_type	object_id	property_id	property_data
1 byte 0x02 or 0x03	1 byte 0x01	2 bytes	4 bytes	2 bytes	N bytes or 2 bytes

service_flags : flags_response = 1, error= 0 or 1

service_id : 0x01 for READ_PROPERTY

object_type : same as the request

object_id : same as the request

property_id : same as the request

property_data : If error in service_flags is 0, the value of the property with the number of bytes of its type. If not, 2 bytes of type ERROR identifying the error code.

4.2.2 WRITE_PROPERTY service

This service is used to write an object's property.

The DTE sends a request frame with the following frame_data:

service_flags	service_id	object_type	object_id	property_id	property_data
1 byte 0x00	1 byte 0x02	2 bytes	4 bytes	2 bytes	n byte

service_flags : is_response = 0, error = 0

service_id : 0x02 for WRITE_PROPERTY

object_type : the object type identifier, defined later in this document

object_id : the object identifier, specific to each object type, i.e. two objects with different types can have the same id

property_id : identify the property in the object

property_data : the data in the right data type.

The Xcom-232i responds with a frame with the following frame_data:

service_flags	service_id	object_type	object_id	property_id	property_data
1 byte 0x02 or 0x03	1 byte 0x02	2 bytes	4 bytes	2 bytes	0 or 2 bytes

service_flags : flags_response = 1, error= 0 or 1

service_id : 0x01 for READ_PROPERTY

object_type : same as the request

object_id : same as the request

property_id : same as the request

property_data : If error in service_flags is 0, 0 byte of data. If not, 2 bytes of type ERROR identifying the error code.

4.2.3 Format

The property data are encoded in different formats described below. Some properties have a format that can be different from one object to an other for the same object_type. For example an the value_qsp of parameter can be an ENUM or a FLOAT depending on the parameter id (identified by the object_id). In this case it is described here as type DYNAMIC. The DTE must then know the exact type of the property for each object to decode it.

BOOL : binary data, 1 byte, 0 = false, 1 = true, other values are invalid

FORMAT : a property what define the format of an other property, 16 bit integer

SHORT_ENUM : a value that is part of a enumeration of possible values, represented with a 16 bit integer

LONG_ENUM : a value that is part of a enumeration of possible values, represented with a 32 bit integer

ERROR : 16 bit error code

INT32 : 32 bit signed value

FLOAT : float in 32 bit IEEE 754 format, little endian

STRING : ISO_8859-15 string of 8 bit characters

DYNAMIC : a property with a different format for each object id

BYTE_STREAM : a stream a byte of abitrary length

example of dynamic property:

for the object type 1 and object id 3000 (XT batery voltage), the format is FLOAT and “value” is a 4 byte IEEE 754 little endian float.

4.3 Error codes

The following error codes of type ERROR can be returned:

name	**error_id**	**meaning**
INVALID_FRAME	0x0001	malformed frame
DEVICE_NOT_FOUND	0x0002	wrong dst_addr field
RESPONSE_TIMEOUT	0x0003	no response of the server
SERVICE_NOT_SUPPORTED	0x0011	wrong service_id field
INVALID_SERVICE_ARGUMENT	0x0012	wrong service_data
SCOM_ERROR_GATEWAY_BUSY	0x0013	gateway (for example XCOM-232i) busy
TYPE_NOT_SUPPORTED	0x0021	the object_type requested doesn't exist
OBJECT_ID_NOT_FOUND	0x0022	no object with this object_id was found
PROPERTY_NOT_SUPPORTED	0x0023	the property identified by property_id doesn't exist
INVALID_DATA_LENGTH	0x0024	the field property_data has an invalid number of bytes
PROPERTY_IS_READ_ONLY	0x0025	a writing to this property is not allowed
INVALID_DATA	0x0026	this value is impossible for this property
DATA_TOO_SMALL	0x0027	the value is below the minimum limit
DATA_TOO_BIG	0x0028	the value is above the maximum limit
WRITE_PROPERTY_FAILED	0x0029	writing is possible, but failed
READ_PROPERTY_FAILED	0x002A	reading is possible, but failed
ACCESS_DENIED	0x002B	insufficient user access

SCOM_ERROR_OBJECT_NOT_SUPPORTED	0x002C	this object id, through existant, is not supported by the current implementation of the gateway
SCOM_ERROR_MULTICAST_READ_NOT_SUPPORTED	0x002D	Read operation is not supported when used on multicast adresses.
OBJECT_PROPERTY_INVALID	0x002E	During a file transfer, the use of this property was unexpected
FILE_OR_DIR_NOT_PRESENT	0x002F	Attempt to download a file not present on the sd card
FILE_CORRUPTED	0x0030	A read error ocurred during the download of a file
INVALID_SHELL_ARG	0x0081	the command line tool used received the wrong arguments

Remark: when writing Xtender parameters, take in acount the parameters interdependencies that reduce the allowed value ranges. See in RCC manual : "APPENDIX 1: LIST OF CONFIGURATION INTERDEPENDENCIES".

4.4 User info objects

These objects are the information about the current state of the system. They cannot be modified and their values change during the operation of the system. Previously known as system states.

object_type = 0x0001

object_id : see the table in next section

4.4.1 Properties

Name	property_id	format	remark
Value	0x0001	DYNAMIC	variable length, see the format in following table

4.4.2 Available user info

The available user information is the same as the values that can be chosen to be displayed on the RCC. The user information is related with the inverter parameters that can be configured with the RCC. The functionalities of each parameter are described in the RCC manual. You can easily find specific parameters by using the parameter index at the end of the manual.

4.4.3 Software version encoding

The software version is of the form X.Y.Z. It is encoded in a 32 bit unsigned value:

8 bit (MSB): X	8 bit: reserved	8 bit: Y	8 bit (LSB):Z

The 32 bit value is formed by combining the SOFT ID MSB (Most Significant Bits) and SOFT ID LSB (Least Significant Bits) User Info. These two values are in FLOAT and must be converted to 16 unsigned beforehand.

4.4.4 FID encoding

The FID is a unique identifier for the device. It is encoded in a 32 bit unsigned value.

This value is formed by combining the ID FID MSB and ID FID LSB User Info. These two values are in FLOAT and must be converted to 16 unsigned beforehand.

4.5 Parameter objects

object_type = 0x0002

All parameters accessible from the remote control can also be modified with the protocol. The behaviour is the same as if a physical person changes the value with the remote control buttons. Currently, only changes at the level qsp are possible.

Values of type FLOAT are stored internally in various 16 bit fixed point formats. For this reason, the read back value after a write can be rounded slightly.

4.5.1 Properties

Name	property_id	format	Remark
value_qsp	0x0005	DYNAMIC	the value that can be entered on the remote control in level qsp or installer.
min_qsp	0x0006	DYNAMIC	Minimum that can be entered on the remote control in level qsp or installer.
max_qsp	0x0007	DYNAMIC	Maximum that can be entered on the remote control in level qsp or installer.
level_qsp	0x0008	ENUM	accessibility level of this parameter modifiable in level qsp or installer.
unsaved_value_qsp	0x000D	DYNAMIC	the value that can be entered on the remote control in level qsp or installer, but without saving the value in internal flash. See section "Cyclic write of parameters on the Xtender Inverter" for more detail.

4.5.2 Values of level properties

The property `level_qsp` of type ENUM can take the following values:

Name	value
VIEW_ONLY	0x0000
BASIC	0x0010
EXPERT	0x0020
INSTALLER	0x0030
QSP	0x0040

4.5.3 Available parameters on the Xtender Inverter

The change of parameters when the inverters are in operation should be done carefully. The modification of parameters can restart the corresponding algorithm inside the inverter. For example, the change of a delay can restart the timer attached to it.

object_id : a number starting at 1000. See the complete parameter references at the end of the RCC User manual.

4.5.4 Cyclic write of parameters on the Xtender Inverter

The Xtender inverter store the parameter values in a non volatile flash memory. Because of the endurance of this memory, the number of write on a single parameter property is only guaranteed for 1000 write operations.

To allow the cyclic write of parameters without count limit, the flash should not be accessed. There are two different ways to archive this. The first is to deactivate the write via parameter {1550} "Parameters saved in flash memory". On newer software versions, there is the possibility to modify the value without accessing the flash with the property *unsaved_value_qsp*.

Deactivating the save in flash with {1550}

The parameter {1550} "Parameters saved in flash memory" has the value "yes" by default. A write of "no" to this parameter value stop the write in the non-volatile flash memory. This operation is written in the flash memory only the first time, so consecutive writes of the value "no" to {1550} can be repeated without limit.

After parameter {1550} has been set to "no", all other parameters can be written without count limit. Because the values of all other parameters are not stored in flash, the read operation will give the values before {1550} as be changed to "no". Also, after a reset the old values will be taken.

To use the inverter with cyclic write operations you must:

- ensure that all inverters have a firmware version >= 1.4.6
- set the parameter {1550} to "no" on all targeted inverter
- avoid to write cyclically on other devices like BSP, RCC, ...
- ensure that no "reset default/factory settings", "apply configuration file (masterfile)" or modification with the remote control change {1550} to "yes"

It is a good pratice to cyclically write "no" to {1550}.

A write of "yes" to the parameter {1550} reactivate the write in flash. It will be written in the flash every time and should not be used more that 1000 time.

Use of the *unsaved_value_qsp* property

This possibility is only available since version XT >= 1.6.12.

The property *unsaved_value_qsp*, described in section "Parameter objects, Properties", allows to modify the value used by the device software in RAM, without changing the value saved in flash and loaded at start up. The property *value_qsp* can be used at the same time, for example to configure the default value after a turn off or a watchdog reset.

4.5.5 Cyclic write of parameters on VarioTrack and VarioString

The Variotrack and Variostring behave the same way as the Xtender Inverter. The parameter {10058} for the VarioTrack and {14069} for the VarioString allows to deactivate the write in non volatile memory.

The property *unsaved_value_qsp* is available since version VT >= 1.6.14 and VS >= 1.6.12.

4.5.6 Cyclic write of parameters on other devices

On other devices, such as BSP, Xcom-232i, Xcom-CAN, there no parameter to deactivate the writing to the non volatile flash memory.

On BSP >= 1.6.12, xcom-CAN >= 1.6.12 and Xcom-232i >= 1.6.12, the property *unsaved_value* can be used for cyclical write.

4.5.7 Hours encoding

the hours encoding is in minute since 00:00 in INT32. For example 13:41 is 13*60+41 = 821.

4.5.8 Days of the week encoding

The days of the week selection (parameters {1205}, for example) is coded as a bit field in a INT32. A day selected as it bit set to 1.

bit	BIT31-7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
day of the week	undefined	SU	SA	FR	TH	WE	TU	MO

4.5.9 Month of the year encoding

The month of the year selection (parameters {1479}, for example) is coded as a bit field in a INT32. A month selected as it bit set to 1. January is BIT0 and December BIT11. The BIT31 to 12 are undefined.

4.5.10 Date and time encoding

The time of the Real Time Clock of the system is coded as a INT32. The value is the number of second since 1.1.1970 00:00:00. The parameters {5002} (Date) take and return the value that contains the complete day and hour information.

4.5.11 Signal encoding

The Signal (parameters {1468}, for example) is coded as a INT32. To send a signal, you must write the value 1 to the parameter value.

4.6 Message objects

The Message objects are supported for Xcom-232 with version >= 1.5.0.

The messages sent by the devices on the communication bus are stored by Xcom-232i in its non-volatile flash memory. They can be read on the Xcom-232i (address 501) later.

4.6.1 Description of the reading function:

Reading a message with index 0 will return the last saved message in the flash memory of the Xcom-232i. In the reponse frame from the SCOM the first data indicates the number of remaining messages before attaining the very first message saved in the flash memory (this behaviour is identical with the history of RCC). A pointer is saved in the Xcom-232i when reading the index 0 (SCOM_MSG_IDX).

Reading a message with an index superior to 0 will return the message saved in the SCOM_MSG_IDX index.

Reading a message with index 0 will erase the flag informing that there are new messages.

If a new message is received after the last reading of index 0, the notification flag is reactivated. Then the PC user must make a new index 0 reading in order to update the pointer (SCOM_MSG_IDX).

4.6.2 Notification of new messages:

A notification flag indicating new messages is sent in every response frame from the Xcom.

4.6.3 Sorting the messages

In order to determine whether the PC user has received all messages, he has to create a unique identifier including the address of the source, the time and date.

4.6.4 Request frame :

The server sends a request in the following format :

flags	service_id	object_type	object_id	property_id
1 byte 0x00	1 byte 0x01	2 bytes	4 bytes	2 bytes

flags : is_response =0, error=0

service_id : 0x01 for READ_PROPERTY

object_type : 0x0003 (MESSAGE)

object_id : Index desired message

property_id : 0x00

4.6.5 Response frame :

The XCOM-232i responds with the following format :

service_flags	service_id	object_type	object_id	property_id	property_data
1 byte 0x02 or 0x03	1 byte 0x01	2 bytes	4 bytes	2 bytes	N bytes

flags :	is_response = 1, error= 0 ou 1
service_id :	Same value as the query
object_type :	Same value as the query
object_id :	Same value as the query
property_id :	Same value as the query
property_data :	If error=0 → the message asked (see 4.6.6)
	If error=1 → 2 bytes for the error code

4.6.6 Content of property_data

Name	Size	Format	Remark
message_total_number	4 bytes	INT32	The total number of message in the XCOM-232i
message_type	2 byte	ENUM	The number defining the meaning of the message. See the appendix part messages.
source_address	4 byte	INT32	Source address of the message. See 3.5 Addressing the devices.
timestamp	4 byte	INT32	The time at which the message occurred in seconds since January 1, 1970.
value	4 byte	DYNAMIC	An optional value of the message. Not yet used currently.

4.7 Custom datalog field object

object_type = 0x0005

The custom datalog field is a string present in each Xcom-232i on the system. It has a maximum length of 250 bytes and is stored in volatile memory. The most recent string of all the Xcom-232i in a system is stored at the end of the datalog file when it is written to the SD card. This appends every day at 00:00:30 or when the user requests it via parameter {5059}, for example.

During the save process of the custom datalog field on the SD card, the use of the SCOM service results in an error “0x0013 SCOM_ERROR_GATEWAY_BUSY”.

After a reset, this value is an empty string. Only the service WRITE_PROPERTY on property_id = 0x1 is allowed.

4.7.1 Format of the string

The 3 characters ; , and # are reserved characters. The characters ; and , are used as a separator within the string.

The string should begin with a string identifier, for example #STU for the company Studer. The # character is used to identify the name of the string.

String example:

"#STU:,Service datalog text,,STUDER INNOTEC SA,Rue des Casernes 57,1950 Sion,Switzerland,,Tel.: +41 (0)27 205 60 80,Fax.: +41 (0)27 205 60 88,info(at)studer-innotec.com,,Long:46.227649,Lat:7.380954".

4.7.2 Importing into Data Analysis Tool

The imported data is displayed on the second sheet of the Data Analysis Tool.

The string identifier is not shown in the display of the Data Analysis Tool.

Each ; or , of the string corresponds to a line break in the Data Analysis Tool.

4.7.3 Property

Name	**property_id**	**format**	**Remark**
`value`	0x0001	STRING	A string of maximum 250 bytes that will be stored in the datalog file. This value is write only.

4.8 File transfer object

This object allows to download the files stored on the SD card of the Xcom-232i. The object_type defines the accessed directory. File names are stripped of their directory name. The current version only supports the datalog directory.

4.8.1 Request frame

The server sends a request in the following format:

flags	service_id	object_type	object_id	property_id	property_data
1 byte 0x00	1 byte 0x01	2 bytes	4 bytes	2 bytes	N bytes

flags: is_response = 0, error = 0

service_id: 0x01 for READ_PROPERTY

0x02 for WRITE_PROPERTY

object_type: 0x0101 (Datalog Transfer, content of CSVFILES\LOG)

object_id: 0x00000001 → Directory list

0x00000002 → File access

property_id: 0x0000 → Invalid Action

0x0021 → SD_Start

0x0022 → SD_Datablock

0x0023 → SD_Ack_Continue

0x0024 → SD_Nack_Retry

0x0025 → SD_Abort

0x0026 → SD_Finish

property_data: If object_id = Directory list → No data

If property_id = Start → filename "LGYYMMDD.CSV"[1] in Big Endian

If property_id = Ack and Continue → No data

If property_id = Nack and Retry → No data

If property_id = Abort → No data

[1] YY = Year, MM = Month, DD = Day

4.8.2 Response frame

The XCOM-232i responds with the following format:

flags	service_id	object_type	object_id	property_id	property_data
1 byte 0x02	1 byte 0x01	2 bytes	4 bytes	2 bytes	N bytes

flags: is_response = 1, error = 0

service_id: 0x01 for READ_PROPERTY

0x02 for WRITE_PROPERTY

object_type: 0x0101 (Datalog Transfer, content of CSVFILES\LOG)

object_id: 0x00000001 → Directory list

0x00000002 → File access

property_id: 0x0000 → Invalid Action

0x0021 → SD_Start

0x0022 → SD_Datablock

0x0023 → SD_Ack_Continue

0x0024 → SD_Nack_Retry

0x0025 → SD_Abort

0x0026 → SD_Finish

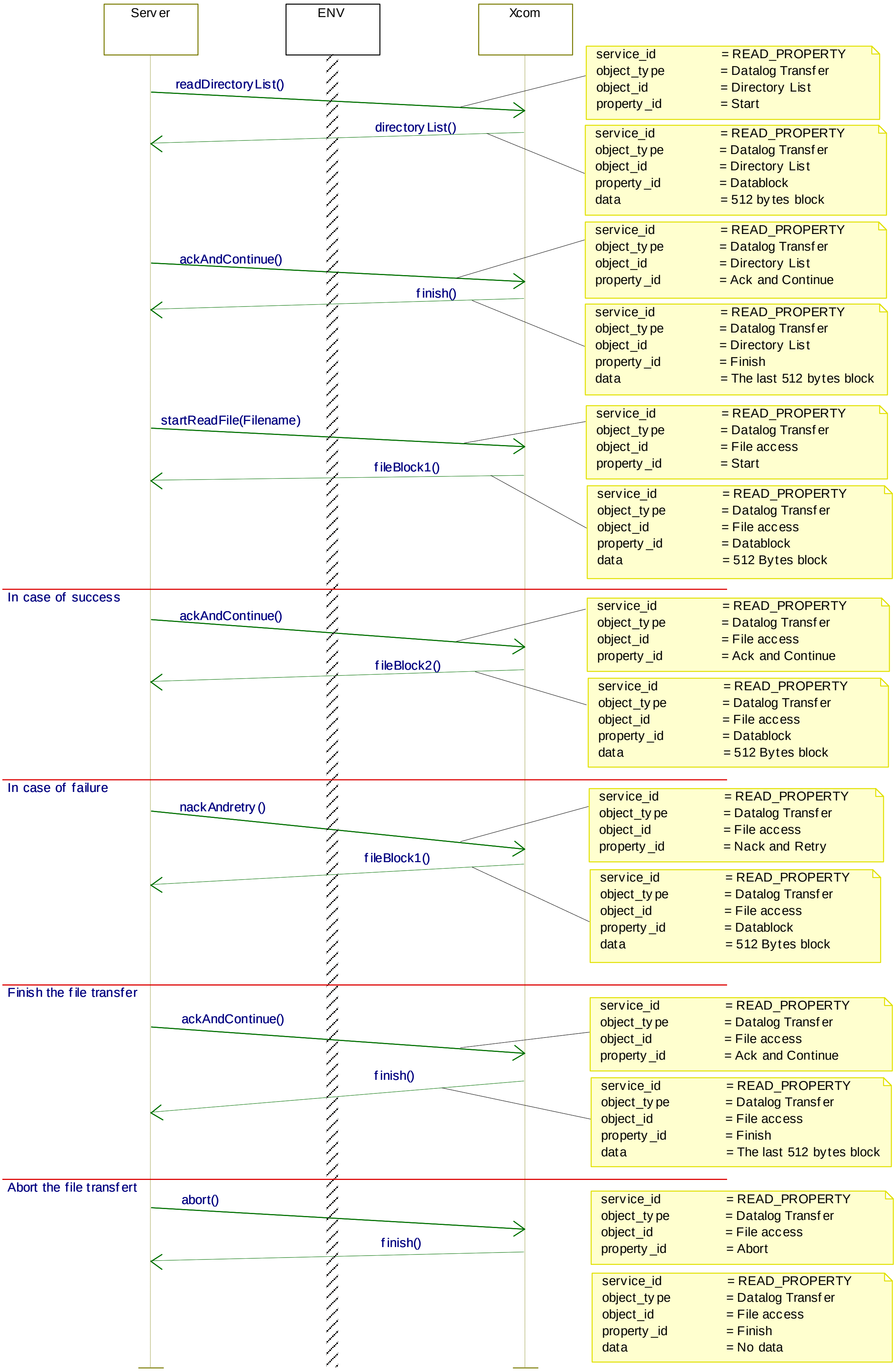
property_data: If object_id = Directory list → Each filename with "CR" for separation

If property_id = Datablock → 512 Bytes / block

If property_id = Finish → No data

If error = 1 → Error number

4.8.3 Transfer sequence



Server
ENV
Xcom
readDirectoryList()
service_id = READ_PROPERTY
object_type = Datalog Transfer
object_id = Directory List
property_id = Start
directoryList()
service_id = READ_PROPERTY
object_type = Datalog Transfer
object_id = Directory List
property_id = Datablock
data = 512 bytes block
ackAndContinue()
service_id = READ_PROPERTY
object_type = Datalog Transfer
object_id = Directory List
property_id = Ack and Continue
finish()
service_id = READ_PROPERTY
object_type = Datalog Transfer
object_id = Directory List
property_id = Finish
data = The last 512 bytes block
startReadFile(Filename)
service_id = READ_PROPERTY
object_type = Datalog Transfer
object_id = File access
property_id = Start
fileBlock1()
service_id = READ_PROPERTY
object_type = Datalog Transfer
object_id = File access
property_id = Datablock
data = 512 Bytes block
In case of success
ackAndContinue()
service_id = READ_PROPERTY
object_type = Datalog Transfer
object_id = File access
property_id = Ack and Continue
fileBlock2()
service_id = READ_PROPERTY
object_type = Datalog Transfer
object_id = File access
property_id = Datablock
data = 512 Bytes block
In case of failure
nackAndretry ()
service_id = READ_PROPERTY
object_type = Datalog Transfer
object_id = File access
property_id = Nack and Retry
fileBlock1()
service_id = READ_PROPERTY
object_type = Datalog Transfer
object_id = File access
property_id = Datablock
data = 512 Bytes block
Finish the file transfer
ackAndContinue()
service_id = READ_PROPERTY
object_type = Datalog Transfer
object_id = File access
property_id = Ack and Continue
finish()
service_id = READ_PROPERTY
object_type = Datalog Transfer
object_id = File access
property_id = Finish
data = The last 512 bytes block
Abort the file transfert
abort()
service_id = READ_PROPERTY
object_type = Datalog Transfer
object_id = File access
property_id = Abort
finish()
service_id = READ_PROPERTY
object_type = Datalog Transfer
object_id = File access
property_id = Finish
data = No data

4.9 Multi infos object

This object allows to ask 1 to 76 user infos simultaneously in one request.

4.9.1 Request frame

The DTE sends a request frame with the following frame_data:

service_flags	service_id	object_type	object_id	property_id	property_data
1 byte 0x00	1 byte 0x01	2 bytes	4 bytes	2 bytes	3 * N bytes

service_flags : is_response =0, error=0

service_id : 0x01 for READ_PROPERTY

object_type : 0x000A for Multi Infos

object_id : 0x00000001 for Infos

property_id : 0x0001 (not relevant)

property_data : For each user info to read, 3 bytes of data containing :

Data	Format	Descriptions
0-1	uint16_t	User info reference
2	uint8_t	Aggregation type

The possible values for the aggragation type are :

- 0x00 : read only the value of the master device.
- 0x01 - 0x0F : read only the value of the device with uid 0x01 to 0x0F.
- 0x10 - 0xFC : reserved.
- 0xFD : read the average value for all devices of the same type.
- 0xFE : sum of values for all devices of the same type.
- 0xFF : reserved.

4.9.2 Response frame

The Xcom-232i responds with a frame with the following frame_data:

service_flags	service_id	object_type	object_id	property_id	property_data
1 byte 0x02 or 0x03	1 byte 0x01	2 bytes	4 bytes	2 bytes	2 bytes error or 8 + 7 * N bytes

service_flags : flags_response = 1, error= 0 or 1

service_id : 0x01 for READ_PROPERTY

object_type : same as the request

object_id : same as the request

property_id : same as the request

property_data : If error in service_flags is 1, 2 bytes of type ERROR identifying the error code. Otherwise, the following data structure :

Data	Format	Descriptions
0-3	uint32_t	32 bits data with the following structure : • Bits 0, 1, 2, 3 : version of the service in Xcom-LAN/GSM. 1st version is 0000b • Bit 4: 0=Xcom-LAN, 1=Xcom-GSM • Bit 5: 0=no XT, 1=XT present • Bit 6: 0=no BSP, 1=BSP present • Bit 7: 0=no VT, 1=VT present • Bit 8: 0=no VS, 1=VS present • Bits 9 à 31 : reserved
4-7	uint32_t	Date/time posix on 4 bytes
8-…	uint16_t	User info reference (similar to request frame)
	uint8_t	Aggregation type (similar to request frame)
	float32_t	Value of the requested user info, as a 4-bytes float.

5. Examples of frames

The byte stream is represented in hexadecimal and the encoding is little endian as specified in chapter 3.2.

In the different exemples, the “frame_flags” byte contains the value 0x34, 00110100b in binary code. It means :

- BIT2 : SD card present.
- BIT4 : there is a new datalog file on the SD card.
- BIT5 : the datalogger is supported.

5.1 C library

A portable C library that implements the protocol is included with this specification on www.studer-innotec.com. See the documentation provided with the library for more detail.

5.2 Command line tool

To help the implementation of the protocol we supply also the command line tool `scom.exe`. It is included with the protocol specification available on www.studer-innotec.com.

5.3 Read the value of a user info

Generated by the command:

```
>scom.exe  --port=COM3  --verbose=3  read_property  src_addr=1  dst_addr=101
object_type=1 object_id=3000 property_id=1 format=FLOAT
```

Request

start_-byte	frame_-flags	src_addr	dst_addr	data_length	header_-checksum	frame_-data	data_-checksum
1 byte	1 byte	4 bytes	4 bytes	2 bytes	2 bytes	N bytes	2 bytes
always AA	0	1	101 (first inverter)	= N = 10	computed	10 bytes	computed
AA	00	01 00 00 00	65 00 00 00	0A 00	6F 71	10 bytes	C5 90

service_flags	service_id	object_type	object_id	property_id	property_data
1 byte	1 byte	2 bytes	4 bytes	2 bytes	0 byte
is_response=false error=false	read_property	user_info	3000 (battery voltage)	value	-
00	01	01 00	B8 0B 00 00	01 00	-

Total number of bytes: 14+10+2 = 26 bytes

Response

start_-byte	frame_-flags	src_addr	dst_addr	data_length	header_-checksum	frame_-data	data_-checksum
1 byte	1 byte	4 bytes	4 bytes	2 bytes	2 bytes	N bytes	2 bytes
always AA	00110100	101 (first inverter)	1	= N = 14	computed	14 bytes	computed
AA	34	65 00 00 00	01 00 00 00	0E 00	A7 45	14 bytes	0D CB

service_flags	service_id	object_type	object_id	property_id	property_data
1 byte	1 byte	2 bytes	4 bytes	2 bytes	4 bytes
is_response = true error = false	read_property	user_info	3000 (battery voltage)	value	12.3594
02	01	01 00	B8 0B 00 00	01 00	00 C0 45 41

Total number of bytes: 14+14+2 = 30 bytes

5.4 Read the qsp_value of a parameter

Generated by the command:

```
> scom.exe --port=COM3 --verbose=3 read_property src_addr=1 dst_addr=101
object_type=2 object_id=1138 property_id=5 format=FLOAT
```

Request

start_-byte	**frame_-flags**	**src_addr**	**dst_addr**	**data_length**	**header_-checksum**	**frame_-data**	**data_-checksum**
1 byte	1 byte	4 bytes	4 bytes	2 bytes	2 bytes	N bytes	2 bytes
always AA	0	1	101 (first inverter)	= N = 10	computed	10 bytes	computed
AA	00	01 00 00 00	65 00 00 00	0A 00	6F 71	10 bytes	7D D9

service_flags	**service_id**	**object_type**	**object_id**	**property_id**	**property_data**
1 byte	1 byte	2 bytes	4 bytes	2 bytes	0 bytes
is_response = false error = false	read_property	parameter	1138 (Battery charge current)	value_qsp	-
00	01	02 00	72 04 00 00	05 00	-

Total number of bytes: 14+10+2 = 26 bytes

Response

start_-byte	**frame_-flags**	**src_addr**	**dst_addr**	**data_length**	**header_-checksum**	**frame_-data**	**data_-checksum**
1 byte	1 byte	4 bytes	4 bytes	2 bytes	2 bytes	N bytes	2 bytes
always AA	00110100	101 (first inverter)	1	= N = 14	computed	14 bytes	computed
AA	34	65 00 00 00	01 00 00 00	0E 00	A7 45	14 bytes	31 0B

service_flags	**service_id**	**object_type**	**object_id**	**property_id**	**property_data**
1 byte	1 byte	2 bytes	4 bytes	2 bytes	4 bytes
is_response = true error = false	read property	parameter	1138 (battery charge current)	value_qsp	60
02	01	02 00	72 04 00 00	05 00	00 00 70 42

Total number of bytes: 14+14+2 = 30 bytes

5.5 Write the qsp_value of a parameter

Set the battery charge current at 12.0 A. Generated by the command:

```
>scom.exe  --port=COM3  --verbose=3  write_property  src_addr=1  dst_addr=101
object_type=2 object_id=1138 property_id=5 format=FLOAT value=12.0
```

Request

start_-byte	**frame_-flags**	**src_addr**	**dst_addr**	**data_length**	**header_-checksum**	**frame_-data**	**data_-checksum**
1 byte	1 byte	4 bytes	4 bytes	2 bytes	2 bytes	N bytes	2 bytes
always AA	0	1	101 (first inverter)	= N = 14	computed	14 bytes	computed
AA	00	01 00 00 00	65 00 00 00	0E 00	73 79	14 bytes	FF 9B

service_flags	**service_id**	**object_type**	**object_id**	**property_id**	**property_data**
1 byte	1 byte	2 bytes	4 bytes	2 bytes	4 bytes
is_response = false error = false	write_property	parameter	1138 (Battery charge current)	value_qsp	12.0
00	02	02 00	72 04 00 00	05 00	00 00 40 41

Total number of bytes: 14+14+2 = 30 bytes

Response

start_-byte	**frame_-flags**	**src_addr**	**dst_addr**	**data_length**	**header_-checksum**	**frame_-data**	**data_-checksum**
1 byte	1 byte	4 bytes	4 bytes	2 bytes	2 bytes	N bytes	2 bytes
always AA	00110100	101 (first inverter)	1	= N = 10	computed	10 bytes	computed
AA	34	65 00 00 00	01 00 00 00	0A 00	A3 3D	10 bytes	80 F6

service_flags	**service_id**	**object_type**	**object_id**	**property_id**	**property_data**
1 byte	1 byte	2 bytes	4 bytes	2 bytes	0 byte
is_response = true error = false	write_property	parameter	1138 (battery charge current)	value_qsp	-
02	02	02 00	72 04 00 00	05 00	-

Total number of bytes: 14+10+2 = 26 bytes

5.6 Read messages

Reading of message 0. Generated by the command:

```
>scom --port=COM3 read_property src_addr=1 dst_addr=501 object_type=3
object_id=0 property_id=0 format=BYTE_STREAM
```

Request

start_-byte	frame_-flags	src_addr	dst_addr	data_length	header_-checksum	frame_-data	data_-checksum
1 byte	1 byte	4 bytes	4 bytes	2 bytes	2 bytes	N bytes	2 bytes
always AA	0	1	501 (Xcom-232i gateway)	= N = 10	computed	10 bytes	computed
AA	00	01 00 00 00	F5 01 00 00	0A 00	00 D6	10 bytes	03 17

service_flags	service_id	object_type	object_id	property_id	property_data
1 byte	1 byte	2 bytes	4 bytes	2 bytes	0 byte
is_response = false error = false	read_property	message	0 (message 0)	0	-
00	01	03 00	00 00 00 00	00 00	-

Total number of bytes: 14+10+2 = 26 bytes

Response

start_-byte	frame_-flags	src_addr	dst_addr	data_length	header_-checksum	frame_-data	data_-checksum
1 byte	1 byte	4 bytes	4 bytes	2 bytes	2 bytes	N bytes	2 bytes
always AA	00110100	501 (Xcom-232i gateway)	1	= N = 28	computed	28 bytes	computed
AA	34	F5 01 00 00	01 00 00 00	1C 00	46 0A	28 bytes	77 60

service_flags	service_id	object_type	object_id	property_id	property_data
1 byte	1 byte	2 bytes	4 bytes	2 bytes	18 bytes
is_response = true error = false	read_property	message	0 (message 0)	0	-
02	01	03 00	00 00 00 00	00 00	D0 03 00 00 10 00 69 00 00 00 82 AD 9E 59 00 00 00 00

Total number of bytes: 14+28+2 = 44 bytes

« Property_data » decoding :

Décodage de la partie property_data :

Name	Size	Value	Format	Remark
message_total_number	4 bytes	D0 03 00 00 = 976d	INT32	The total number of message in the XCOM-232i
message_type	2 bytes	10 00 = 16d : Warning (016): Fan error detected	ENUM	The number defining the meaning of the message.
source_address	4 bytes	69 00 00 00 = 105d : Xtender 5	INT32	Source address of the message. See 3.5 Addressing the devices.
timestamp	4 bytes	82 AD 9E 59 = 1503571330 : 24.08.2017 at 10:42:10	INT32	The time at which the message occurred in seconds since January 1, 1970.
value	4 bytes	00 00 00 00	DYNAMIC	An optional value of the message. Not yet used currently.

5.7 File transfer object

5.7.1 Reading a directory

Generated by the command:

```
>scom --port=COM3 read_property src_addr=1 dst_addr=501 object_type=257
object_id=1 property_id=33 format=BYTE_STREAM
```

Request

start_byte	frame_flags	src_addr	dst_addr	data_length	header_checksum	frame_data	data_checksum
1 byte	1 byte	4 bytes	4 bytes	2 bytes	2 bytes	N bytes	2 bytes
always AA	0	1	501 (Xcom-232i gateway)	= N = 10	computed	10 bytes	computed
AA	00	01 00 00 00	F5 01 00 00	0A 00	00 D6	10 bytes	24 56

service_flags	service_id	object_type	object_id	property_id	property_data
1 byte	1 byte	2 bytes	4 bytes	2 bytes	0 byte
is_response = false error = false	read_property	file tranfer	directory list	0x0021 start	-
00	01	01 01	01 00 00 00	21 00	-

Total number of bytes: 14+10+2 = 26 bytes

Response

start_byte	frame_flags	src_addr	dst_addr	data_length	header_checksum	frame_data	data_checksum
1 byte	1 byte	4 bytes	4 bytes	2 bytes	2 bytes	N bytes	2 bytes
always AA	00100111	501 (Xcom-232i gateway)	1	= N = 36	computed	36 bytes	computed
AA	27	F5 01 00 00	01 00 00 00	24 00	41 8B	36 bytes	EE C9

service_flags	service_id	object_type	object_id	property_id	property_data
1 byte	1 byte	2 bytes	4 bytes	2 bytes	26 bytes
is_response = true error = false	read_property	file tranfer	directory list	0x0026 finish	LG171010.CSV LG171011.CSV
02	01	01 01	01 00 00 00	26 00	4C 47 31 37 31 30 31 30 2E 43 53 56 0A 4C 47 31 37 31 30 31 31 2E 43 53 56 0A

Total number of bytes: 14+36+2 = 52 bytes

5.7.2 Reading a datalog file

Generated by the command:

```
>scom --port=COM3 read_property src_addr=1 dst_addr=501 object_type=257
object_id=2 property_id=33 format=BYTE_STREAM
```

Request

start_-byte	frame_-flags	src_addr	dst_addr	data_length	header_-checksum	frame_-data	data_-checksum
1 byte	1 byte	4 bytes	4 bytes	2 bytes	2 bytes	N bytes	2 bytes
always AA	0	1	501 (Xcom-232i gateway)	= N = 10	computed	10 bytes	computed
AA	00	01 00 00 00	F5 01 00 00	0A 00	00 D6	23 bytes	24 56

service_flags	service_id	object_type	object_id	property_id	property_data
1 byte	1 byte	2 bytes	4 bytes	2 bytes	13 bytes
is_response = false error = false	read_property	file tranfer	file access	0x0021 start	LG171010.CSV
00	01	01 01	02 00 00 00	21 00	4C 47 31 37 31 30 31 30 2E 43 53 56 0A

Total number of bytes: 14+23+2 = 39 bytes

Response

start_-byte	frame_-flags	src_addr	dst_addr	data_length	header_-checksum	frame_-data	data_-checksum
1 byte	1 byte	4 bytes	4 bytes	2 bytes	2 bytes	N bytes	2 bytes
always AA	00100111	501 (Xcom-232i gateway)	1	= N = 36	computed	36 bytes	computed
AA	27	F5 01 00 00	01 00 00 00	24 00	41 8B	36 bytes	EE C9

service_flags	service_id	object_type	object_id	property_id	property_data
1 byte	1 byte	2 bytes	4 bytes	2 bytes	26 bytes
is_response = true error = false	read_property	file tranfer	file access	0	LG171010.CSV LG171011.CSV
02	01	01 01	02 00 00 00	26 00	4C 47 31 37 31 30 31 30 2E 43 53 56 0A 4C 47 31 37 31 30 31 31 2E 43 53 56 0A

Total number of bytes: 14+36+2 = 52 bytes

5.8 Read Multi Infos

Exemple basé sur les 24 demandes pour l'affichage du synoptique sur le portail Xcom.

No	Réf.	Assemblage	Device	Description
1	3000	253: average	XT	Battery voltage [Vdc]
2	3080	254: sum	XT	Energy AC-In from the previous day [kWh]
3	3081	254: sum	XT	Energy AC-In from the current day [kWh]
4	3082	254: sum	XT	Consumers energy of the previous day [kWh]
5	3083	254: sum	XT	Consumers energy of the current day [kWh]
6	3136	254: sum	XT	Output active power [kW]
7	3137	254: sum	XT	Input active power [kW]
8	7000	0: master	BSP	Battery voltage [Vdc]
9	7001	0: master	BSP	Battery current [Adc]
10	7002	0: master	BSP	State of Charge (SoC) [%]
11	7003	0: master	BSP	Power [W]
12	7005	0: master	BSP	BSP Temperature [°C]
13	7007	0: master	BSP	Ah charged today [kAh]
14	7008	0: master	BSP	Ah discharged today [kAh]
15	7009	0: master	BSP	Ah charged yesterday [kAh]
16	7010	0: master	BSP	Ah discharged yesterday [kAh]
17	11000	253: average	VT	Battery voltage [Vdc]
18	11004	254: sum	VT	Power of the PV generator [kW]
19	11007	254: sum	VT	Production in (kWh) for the current day
20	11011	254: sum	VT	Production in (kWh) for the previous day
21	15000	253: average	VS	Battery voltage [Vdc]
22	15010	254: sum	VS	Power of the PV generator [kW]
23	15017	254: sum	VS	Production in (kWh) for the current day
24	15027	254: sum	VS	Production in (kWh) for the previous day

Generated by the command (non implémenté dans scom.exe et la lib):

```
scom --port=COM3 read_property src_addr=1 dst_addr=501 object_type=10
object_id=1 property_id=1 format=BYTE_STREAM
```

Request

start_-byte	frame_-flags	src_addr	dst_addr	data_length	header_-checksum	frame_-data	data_-checksum
1 byte	1 byte	4 bytes	4 bytes	2 bytes	2 bytes	N bytes	2 bytes
always AA	0	1	501 (Xcom-232i gateway)	= N = 10	computed	n bytes	computed
AA	00	01 00 00 00	F5 01 00 00	0A 00	00 D6	n bytes	13 8E

service_flags	service_id	object_type	object_id	property_id	property_data
1 byte	1 byte	2 bytes	4 bytes	2 bytes	76x3=228 bytes
is_response = false error = false	read_property	0x000A=10d Multi Infos	0x00000001 infos	0x0001 value	user ref 2bytes assemblage 1byte
00	01	0A 00	01 00 00 00	01 00	???????

Total number of bytes: min 14+10+3+2 = 29 bytes et max 14+10+76x3+2 = 254 bytes

Response

start_-byte	frame_-flags	src_addr	dst_addr	data_length	header_-checksum	frame_-data	data_-checksum
1 byte	1 byte	4 bytes	4 bytes	2 bytes	2 bytes	N bytes	2 bytes
always AA	00110111	501 (Xcom-232i gateway)	1	= N = 114	computed	n bytes	computed
AA	37	F5 01 00 00	01 00 00 00	72 00	xx xx	n bytes	xx xx

service_flags	service_id	object_type	object_id	property_id	property_data
1 byte	1 byte	2 bytes	4 bytes	2 bytes	104 bytes
is_response = true error = false	read_property	Multi Infos	0x00000001 infos	0x0001 value	2 valeurs uint32_t et n valeurs (uitn16_t , uint8_t , float32)
02	01	0A 00	01 00 00 00	01 00	???????

Total number of bytes: min 14+10+2x4+7+2 = 41 bytes et max 14+10+2x4+76x7+2 = 566 bytes.

6. Other services

6.1 GUID object

6.1.1 Description of the reading function:

The GUID is used for remotely identifying the installation. This function is only for the Modem or Ethernet mode.

6.1.2 Request frame :

The server sends a request in the following format :

flags	service_id	object_type	object_id	property_id
0x00	0x01	2 bytes	4 bytes	2 bytes

flags : is_response =0, error=0

service_id : 0x01 for READ_PROPERTY

object_type : 0x04 (GUID)

object_id : 0x00 → Not used

property_id : 0x00 → Not used

6.1.3 Response frame :

The XCOM-232i responds with the following format :

service_flags	service_id	object_type	object_id	property_id	property_data
0x02 or 0x03	0x01	2 bytes	4 bytes	2 bytes	N bytes

flags : is_response = 1, error= 0 ou 1

service_id : Same value as the query

object_type : Same value as the query

object_id : Same value as the query

property_id : Same value as the query

property_data : If error=0 → the global unique identifier (GUID) (see 4.7.4)

If error=1 → 2 bytes for the error code

6.1.4 Content of data property

Name	Size	Format	Remark
GUID	16 bytes	INT32	The Global Unique Identifier (GUID) in 16-bytes format in little Endian.

6.1.5 Read GUID sample

Reading of GUID frame.

Request

start_-byte	**frame_-flags**	**src_addr**	**dst_addr**	**data_length**	**header_-checksum**	**frame_-data**	**data_-checksum**
1 byte	1 byte	4 bytes	4 bytes	2 bytes	2 bytes	N bytes	2 bytes
always AA	0	1	501 (Xcom-232i gateway)	= N = 10	computed	10 bytes	computed
AA	00	01 00 00 00	F5 01 00 00	0A 00	00 D6	10 bytes	04 1F

service_flags	**service_id**	**object_type**	**object_id**	**property_id**	**property_data**
1 byte	1 byte	2 bytes	4 bytes	2 bytes	0 byte
is_response = false error = false	read_property	GUID	0	0	-
00	01	04 00	00 00 00 00	00 00	-

Total number of bytes: 14+10+2 = 26 bytes

Response

start_-byte	**frame_-flags**	**src_addr**	**dst_addr**	**data_length**	**header_-checksum**	**frame_-data**	**data_-checksum**
1 byte	1 byte	4 bytes	4 bytes	2 bytes	2 bytes	N bytes	2 bytes
always AA	00110100	501 (Xcom-232i gateway)	1	= N = 26	computed	26 bytes	computed
AA	23	F5 01 00 00	01 00 00 00	1A 00	33 4B	26 bytes	34 A3

service_flags	**service_id**	**object_type**	**object_id**	**property_id**	**property_data**
1 byte	1 byte	2 bytes	4 bytes	2 bytes	16 bytes
is_response = true error = false	read_property	GUID	0	0	GUID
02	01	04 00	00 00 00 00	00 00	3A D0 1B 0B 58 C9 32 BA EC 4D 35 28 61 3D 7D 40

Total number of bytes: 14+26+2 = 42 bytes

6.2 Screen object

Modem connection with the Xtender installation is made by remote control. The application on the server must send the state of the 4 keys and will in turn receive the resulting display. ~~A distinction is made between pressing and holding the button down (non implémenté)~~.

The display on the Xcom-232i is built in a buffer. In order to make the system more responsive, it’s important to reduce the quantity of returned data as much as possible.

6.2.1 Request frame :

The server sends a request in the following format :

flags	service_id	object_type	object_id	property_id
0x00	0x01	2 bytes	4 bytes	2 bytes

flags : is_response =0, error=0

service_id : 0x01 for READ_PROPERTY

object_type : 0x100 (SCREEN)

object_id : 0x00 → No buttons pressed (used for refresh)

0x10 → Button "Down" pressed

0x20 → Button "Esc" pressed

0x40 → Button "Set" pressed

0x80 → Button "Up" pressed

property_id : 0x00 → Full screen

~~0x01 → Delta with the last screen (non implémenté)~~

6.2.2 Trame de réponse :

The XCOM-232i responds with the following format :

service_flags	service_id	object_type	object_id	property_id	property_data
0x02 or 0x03	0x01	2 bytes	4 bytes	2 bytes	N bytes

flags : is_response = 1, error= 0 ou 1

service_id : Same value as the query

object_type : Same value as the query

object_id : Same value as the query

property_id : Same value as the query

property_data : If error=0 → the screen (see 4.8.3)

If error=1 → 2 bytes for the error code

6.2.3 Content of data property

The property_id in the request allows to choose a complete image (0x00) or the delta (0x01). The delta of the image is carried out with an XOR in between the previous and the current image.

All displays are then compressed using the following algorithms :

"10" and the number of white bits to send on a total of 14 bits.

"11" and the number of black bits to send on a total of 14 bits.

"0" and varying bits on a total of 15 bits.

BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0	15 successive pixels of different values														
1	0	The number of successive white pixels (0 uptil 16383)													
1	1	The number of successive white pixels (0 uptil 16383)													

6.2.4 Algorithm to decompress images

```
int index = 0;

for(i=0; i < frame_length; i++)
{
        if( buffer[i] & BIT15 )
        {

                int count = buffer[i] & 0x3FFF;
                bool value = buffer[i] & BIT14;

                while(count > 0)
                {
                        setPixel(index, value);
                        index++;
                        count--;
                }
        }
        else
        {
                for(int j = 0; j < 15; j++)
                {
                        bool value = (buffer[i] >> j) & BIT0;
                        setPixel(index, value);
                        index++;
                }
        }
}
```

6.2.5 Read Screen sample

Reading of screen frame.

Request

start_-byte	frame_-flags	src_addr	dst_addr	data_length	header_-checksum	frame_-data	data_-checksum
1 byte	1 byte	4 bytes	4 bytes	2 bytes	2 bytes	N bytes	2 bytes
always AA	0	1	501 (Xcom-232i gateway)	= N = 10	computed	10 bytes	computed
AA	00	01 00 00 00	F5 01 00 00	0A 00	00 D6	10 bytes	01 06

service_flags	service_id	object_type	object_id	property_id	property_data
1 byte	1 byte	2 bytes	4 bytes	2 bytes	0 byte
is_response = false error = false	read_property	screen	up=80 00 00 00 set=40 00 00 00 esc=20 00 00 00 down=10 00 00 00 screen=00 00 00 00	0	-
00	01	00 01	00 00 00 00	00 00	-

Total number of bytes: 14+10+2 = 26 bytes

Response

start_-byte	frame_-flags	src_addr	dst_addr	data_length	header_-checksum	frame_-data	data_-checksum
1 byte	1 byte	4 bytes	4 bytes	2 bytes	2 bytes	N bytes	2 bytes
always AA	00100010	501 (Xcom-232i gateway)	1	= N = 1034	computed	1034 bytes	computed
AA	22	F5 01 00 00	01 00 00 00	0A 04	26 24	1034 bytes	C5 5B

service_flags	service_id	object_type	object_id	property_id	property_data
1 byte	1 byte	2 bytes	4 bytes	2 bytes	1024 bytes
is_response = true error = false	read_property	screen	0	0	screen
02	01	00 01	00 00 00 00	00 00	1024 bytes

Total number of bytes: 14+1034+2 = 1050 bytes