XETHRU
BY NOVELDA

XeThru Module Communication Protocol

Low Level Protocol Documentation

XeThru Application Note **by Novelda AS**

Rev. C - November 06. 2018

Summary

This document describes the serial protocol used by X4 based XeThru sensor modules.

Table of Contents

1 Introduction

XeThru Module Communication Protocol (MCP) is the lowest level communication protocol implementation for the XeThru products. It is used on both the XeThru Module firmware and XeThru host softwares. Novelda provide both sensors and development kits for customer. To avoid confusion, we specify the following definitions in this document:

XeThru Sensor: Sensors that provide solutions for specific applications, i.e. X4M200 for respiration monitoring and X4M300 for presence detection.
XeThru Development Kit: Radar Development Kit provides a reference platform to debug and develop solutions for our customers that want to create their own radar product, i.e. X4M03, X4M06 running open source XeThru Embedded Platform (XEP).
XeThru Module: XeThru sensor + development kit.

XEP is an open source radar development platform for XeThru development kit. All XeThru sensors are also developed based on XEP in addition to custom DSP algorithm. MCP is one important part of XEP source code [1]. Module Connector [2] is the software used to communicate with XeThru modules from a host computer. Module Connector is implemented and distributed as a Shared Object / Dynamic Link Library (DLL) and can be linked in runtime and accessed through an API from several different host environments including MATLAB, Python and C++. MCPW [3] offers a source code implementation of sensor communication and provides examples for different hardware platforms. Typical usage of MCPW is implementing an application on a host platform where Module Connector is not supported, e.g. embedded platforms. MCP is used by both Module Connector and MCPW. This document provides documentation on how to interpret MCP commands from host side software for communication with XeThru modules.

This document will only focus on XeThru Modules based on the X4 chip. For X2 based sensors, please refer to the former version XeThru Serial Protocol [4].

2 Notation

The following notation is used in this document:

```
<X> = Single byte
[X] = Multiple bytes
[“abc”] = [0x61,0x62,0x63] = Ascii text
[X(i)] = 32-bit Integer, 4 bytes
[X(f)] = 32-bit Float, 4 bytes, IEEE 754 format
```

3 Protocol Binary Format

The binary protocol supports two packaging modes, Normal and NoEscape. Normal packaging mode is most common and uses flag bytes and escaping. NoEscape packaging mode does not use escaping, but rather a binary start sequence and packet length. The XeThru serial protocol assumes little-endian byte order.

Normal packaging is used by XeThru sensor application message output: sleep status message, respiration status message, presence status message and so on.

NoEscape packaging is normally used by messages containing larger amount of data, which is used by Generic Float Data Message, Radar Baseband Data Message, Radar Baseband AP Message, PulseDoppler/NoiseMap Message.

3.1 Normal Packaging Mode

Normal packaging mode ensures immediate re-sync of the protocol parsing in the event of discontinuity between the module and host. This way of packaging is therefore considered more robust, in particular when using UART as transport layer. It does however require more processing and is best suited with medium to low data throughput requirement. All high-level module output messages are supported in Normal packaging mode.

Message: `<Start> + [Data] + <CRC> + <End>`

Flag	bytes
<Start>	0x7D
<End>	0x7E
<Esc>	0x7F

3.1.1 Checksum

Checksum is calculated by XOR all bytes from <Start> + [Data] apart from <End>. Note that the CRC is done after escape bytes are removed. This means that CRC is also calculated before adding escape bytes.

3.1.2 Escaping

Escaping means that if an escape byte occurs in a message, the next byte is not <Start>, <End> or <Esc>, but a byte that is part of the message and has same value as one of the flags.

Example: `0x7D + 0x10 + 0x7F + 0x7E + 0x04 + 0xFF + 0x7E`

Here the byte 0x7E in the middle is a part of the message and should not be read as an <End> flag. The 0x7E byte is prepended with the escape byte 0x7F. After parsing for escape and removing start and end bytes, the data becomes:

`0x10 + 0x7E + 0x04 + 0xFF`

3.2 NoEscape Packaging Mode

NoEscape packaging mode is a great advantage when high data throughput is required because of the complexity of adding and subtracting escape bytes is removed. However, in situations where the module and host get out of sync, this method may take longer to resynchronize compared to Normal packaging mode. When using USB this is less likely to be a problem, but when using UART, this should be taken into consideration. The following high throughput messages are using NoEscape packaging mode: Generic Float Data Message, Radar Baseband Data Message, Radar Baseband AP Message, PulseDoppler/NoiseMap Message.

Message: `[XTS_FLAGSEQUENCE_START_NOESCAPE(i)] + [PacketLength(i)] + <RESERVED> + [Data]`

Example: `0x7C + 0x7C + 0x7C + 0x7C + 0x03 + 0x00 + 0x00 + 0x00 + 0x00 + 0x01 + 0x02 + 0x03`

PacketLength is the number of bytes in [Data].

Name	Value
XTS_FLAGSEQUENCE_START_NOESCAPE	0x7C7C7C7C

4 Configuration Procedure

MCP is used by different XeThru products, which means part of its commands just support specific product. All XeThru Modules support radar raw data message output. For XeThru Sensor, it also supports application data message output, such as respiration, presence, movement list and so on. There are different configuration flows depending on developer's needs on data message.

4.1 Get Radar Raw Data from XeThru Module

To get radar raw data from XeThru Module, developer needs to access X4 chip through Manual mode from the module MCU. It allows the host to configure X4 chip settings, including dac_min, dac_max, iterations, pulses_per_step, down conversion, FPS and so on. Please refer to X4 datasheet [5] for introduction on parameter. "Set FPS" command is used to enable or disable radar frame streaming and output. Configuration workflow:

1. Set Module to stop mode
2. Set Module to manual mode to enable direct access to X4
3. Load parameters (pps, dac_min, dac_max, iteration, frame_offset frame_area, etc.)
4. Set FPS to start radar
5. Get radar raw data message

4.2 Get Application Data from XeThru Sensor

At first power-up of the XeThru sensor, the default Profile will be loaded and started with its default User Settings. Each Profile has a default message that the sensor will start sending over the serial communication link. After loading the profile, it can be configured by sending profile configuration commands. Finally, after configuring the profile, the user can send a command to start it. Next time the module is powered, it will automatically load the previous configuration and resume operation. If you want to change sensor behavior, stop loaded profile, reconfigure and start again. Configuration workflow:

1. Set XeThru Sensor to stop mode
2. Load profile
3. Load parameters (detection zone, sensitivity, noise map, led, output, etc.)
4. Set XeThru Sensor to run mode to start profile
5. Get application data message

5 Commands from Host

This section introduces commands from host to configure XeThru modules.

Developer can configure X4 XeThru modules to get radar raw data through X4Driver interface, which provides access to X4 chip configuration, including dac_min, dac_max, iterations, pulses_per_setp, down conversion and FPS. Specially for XeThru sensors like X4M200, X4M300, developer can get application data message output.

5.1 Basic Commands for XeThru Module

5.1.1 Reset module

Use this command to completely reset the sensor module. After the module has responded with ACK, it will wait for 0.5 seconds before the reset procedure starts. This gives the host time to disconnect from the serial connection prior to the module reset, which is necessary when USB interface is used. Module reset is basically the same as power toggling the sensor.

Send: `<Start> + <XTS_SPC_MOD_RESET> + <CRC> + <End> (0x7D 0x22 0x5F 0x7E)`

Response: `<Start> + <XTS_SPR_ACK> + <CRC> + <End> (0x7D 0x10 0x6D 0x7E)`

Protocol codes:

Name	Value
XTS_SPC_MOD_RESET	0x22
XTS_SPR_ACK	0x10

After the module resets, it sends a set of system messages to inform the host about the bootup status. At first startup, the XTS_SPRS_BOOTING message is sent. Then, after the module booting sequence is completed and the module is ready to accept further commands, the XTS_SPRS_READY command is issued.

Response: `<Start> + <XTS_SPR_SYSTEM> + [Responsecode(i)] + <CRC> + <End>`

Protocol codes:

Name	Value
XTS_SPR_SYSTEM	0x30

[Responsecode(i)]:

Name	Value
XTS_SPRS_BOOTING	0x00000010
XTS_SPRS_READY	0x00000011

If the module was in run mode when the reset was called, the module will automatically load and run the previous profile and configuration. If in stop mode, the sensor must be re-configured before started again.

5.1.2 Set baud rate

Set baud rate for serial communication.

Send: <Start> + <XTS_SPC_DIR_COMMAND> + <XTS_SDC_COMM_SETBAUDRATE> + [Baudrate (i)] + <CRC> + <End>

Response: <Start> + <XTS_SPR_ACK> + <CRC> + <End>

Protocol codes:

Name	Value
XTS_SPC_DIR_COMMAND	0x90
XTS_SDC_COMM_SETBAUDRATE	0x80
XTS_SPR_ACK	0x10

[Baudrate(i)]:

Name	Value
XTID_BAUDRATE_9600	9600
XTID_BAUDRATE_19200	19200
XTID_BAUDRATE_38400	38400
XTID_BAUDRATE_115200	115200
XTID_BAUDRATE_230400	230400
XTID_BAUDRATE_460800	460800
XTID_BAUDRATE_500000	500000
XTID_BAUDRATE_576000	576000
XTID_BAUDRATE_921600	921600
XTID_BAUDRATE_1000000	1000000
XTID_BAUDRATE_2000000	2000000
XTID_BAUDRATE_3000000	3000000
XTID_BAUDRATE_4000000	4000000

5.1.3 Ping

The ping command can be used to check the connection to the module and to verify module readiness.

Comes in handy if the XTS_SPRS_READY status message is not possible to receive after reset.

Send: <Start> + <XTS_SPC_PING> + [XTS_DEF_PINGVAL(i)] + <CRC> + <End>

Response: <Start> + <XTS_SPR_PONG> + [Pongval(i)] + <CRC> + <End>

Protocol codes:

Name	Value	Description
XTS_SPC_PING	0x01	Ping command code
XTS_DEF_PINGVAL	0xeeaaeaae	Ping seed value
XTS_SPR_PONG	0x01	Pong response code
XTS_DEF_PONGVAL_READY	0xaaeeaeea	Module is ready
XTS_DEF_PONGVAL_NOTREADY	0xaeeaeeaa	Module is not ready
XTS_DEF_PONGVAL_SAFEMODE	0xffeefeef	Module enter safe mode (after crashing 10 times in a row within 5mins), and will disable all application logic and commands.

5.1.4 Set sensor_mode

X4 based sensors support four kinds of modes:

Mode Name	Value	Description
XTS_SM_RUN	0x01	Only supported by XeThru Sensor, start application profile execution .
XTID_SM_IDLE	0x11	Halts profile execution. Can be resumed by setting mode to Run.
XTS_SM_STOP	0x13	Stop XeThru sensor profile execution.
XTS_SM_Manual	0x12	This mode is supported by all XeThru Modules, support configuring X4 chip at low level and output radar raw data.

Send: `<Start> + <XTS_SPC_MOD_SETMODE> + <Mode> + <CRC> + <End>`

Example:

0x7D 0x20 0x01 0x5C 0x7E (Run mode)

0x7D 0x20 0x13 0x4E 0x7E (Stop mode)

0x7D 0x20 0x12 0x4F 0x7E (Manual mode)

Response: `<Start> + <XTS_SPR_ACK> + <CRC> + <End>`

Protocol codes:

Name	Value
XTS_SPC_MOD_SETMODE	0x20
XTS_SPR_ACK	0x10

5.2 Commands for XeThru Module Raw Data Output Configuration

5.2.1 Set pulses_per_step

Send: `<Start> + <XTS_SPC_X4Driver> + <XTS_SPCX_SET> + [XTS_SPCXI_PULSESPERSTEP (i)] + [Pulsesperstep(f)] + <CRC> + <End>`

Example: `0x7D 0x50 0x10 0x11 0x00 0x00 0x00 0x14 0x38 0x7E` (Sets pulses_per_step to 20)

Response: `<Start> + <XTS_SPR_ACK> + <CRC> + <End>`

Protocol codes:

Name	Value
XTS_SPC_X4Driver	0x50
XTS_SPCA_SET	0x10
XTS_SPCXI_PULSESPERSTEP	0x00000011
XTS_SPR_ACK	0x10

5.2.2 Set iterations

Send: `<Start> + <XTS_SPC_X4Driver> + <XTS_SPCX_SET> + [XTS_SPCXI_ITERATIONS(i)] + [Iterations(i)] + <CRC> + <End>`

Response: `<Start> + <XTS_SPR_ACK> + <CRC> + <End>`

Protocol codes:

Name	Value
XTS_SPC_X4Driver	0x50
XTS_SPCA_SET	0x10
XTS_SPCXI_ITERATIONS	0x00000012
XTS_SPR_ACK	0x10

5.2.3 Set downconversion

Send: `<Start> + <XTS_SPC_X4Driver> + <XTS_SPCX_SET> + [XTS_SPCXI_DOWNCONVERSION (i)] + <Downconversion> + <CRC> + <End>`

Response: `<Start> + <XTS_SPR_ACK> + <CRC> + <End>`

Protocol codes:

Name	Value
XTS_SPC_X4Driver	0x50
XTS_SPCA_SET	0x10
XTS_SPCXI_DOWNCONVERSION	0x00000013
XTS_SPR_ACK	0x10

<Downconversion>:

Value	Comments
0x01	Enable
0x00	Disable

5.2.4 Set dac_min

Send: `<Start> + <XTS_SPC_X4Driver> + <XTS_SPCX_SET> + [XTS_SPCXI_DACMIN(i)] + [Dac_min(i)] + <CRC> + <End>`

Response: `<Start> + <XTS_SPR_ACK> + <CRC> + <End>`

Protocol codes:

Name	Value
XTS_SPC_X4Driver	0x50
XTS_SPCA_SET	0x10
XTS_SPCXI_DACMIN	0x00000016
XTS_SPR_ACK	0x10

5.2.5 Set dac_max

Send: `<Start> + <XTS_SPC_X4Driver> + <XTS_SPCX_SET> + [XTS_SPCXI_DACMAX(i)] + [Dac_max(i)] + <CRC> + <End>`

Response: `<Start> + <XTS_SPR_ACK> + <CRC> + <End>`

Protocol codes:

Name	Value
XTS_SPC_X4Driver	0x50
XTS_SPCA_SET	0x10
XTS_SPCXI_DACMAX	0x00000017
XTS_SPR_ACK	0x10

5.2.6 Set frame_area_offset

Send: <Start> + <XTS_SPC_X4Driver> + <XTS_SPCX_SET> + [XTS_SPCXI_FRAMEAREAOFFSET (i)] + [Offset(f)] + <CRC> + <End>

Response: <Start> + <XTS_SPR_ACK> + <CRC> + <End>

Protocol codes:

Name	Value
XTS_SPC_X4Driver	0x50
XTS_SPCA_SET	0x10
XTS_SPCXI_FRAMEAREAOFFSET	0x00000018
XTS_SPR_ACK	0x10

5.2.7 Set frame_area

Send: <Start> + <XTS_SPC_X4Driver> + <XTS_SPCX_SET> + [XTS_SPCXI_FRAMEAREA(i)] + [Start(f)] + [End(f)] + <CRC> + <End>

Response: <Start> + <XTS_SPR_ACK> + <CRC> + <End>

Protocol codes:

Name	Value
XTS_SPC_X4Driver	0x50
XTS_SPCA_SET	0x10
XTS_SPCXI_FRAMEAREA	0x00000014
XTS_SPR_ACK	0x10

5.2.8 Set fps

Send: <Start> + <XTS_SPC_X4Driver> + <XTS_SPCX_SET> + [XTS_SPCXI_FPS(i)] + [Fps (f)] + <CRC> + <End>

Example:

0x7D 0x50 0x10 0x10 0x00 0x00 0x00 0x00 0x00 0xA0 0x41 0xCC 0x7E (FPS = 20)

0x7D 0x50 0x10 0x10 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x2D 0x7E (FPS = 0)

Response: <Start> + <XTS_SPR_ACK> + <CRC> + <End>

Protocol codes:

Name	Value
XTS_SPC_X4Driver	0x50

Name	Value
XTS_SPCA_SET	0x10
XTS_SPCXI_FPS	0x00000010
XTS_SPR_ACK	0x10

Radar frames are sampled and output when FPS is set to a value greater than zero. If downconversion is enabled, the output message is baseband data. If downconversion is not enabled, the output message is RF data.

5.2.9 Set pin_enable

Set X4 IO pins enable or not.

Send: `<Start> + <XTS_SPC_X4Driver> + <XTS_SPCX_SET> + [XTS_SPCXI_ENABLE(i)] + <Enable> + <CRC> + <End>`

Example: `0x7D 0x50 0x10 0x19 0x00 0x00 0x00 0x01 0x25 0x7E` (Enable)

Response: `<Start> + <XTS_SPR_ACK> + <CRC> + <End>`

Protocol codes:

Name	Value
XTS_SPC_X4Driver	0x50
XTS_SPCA_SET	0x10
XTS_SPCXI_ENABLE	0x00000019
XTS_SPR_ACK	0x10

<Enable>:

Value	Comment
XTID_OUTPUT_CONTROL_DISABLE	0x00
XTID_OUTPUT_CONTROL_ENABLE	0x01

5.2.10 Set iopin_control

The whole procedure to set X4 chip IO pins from host side: Set Pin Enable-> Set IO Pin Control -> Set IO Pin Value.

Send: `<Start> + <XTS_SPC_IOPIN> + <XTS_SPCIOP_SETCONTROL> + [Pin_id(i)] + [Pin_setup(i)] + [Pin_feature(i)] + <CRC> + <End>`

Example: `0x7D 0x40 0x10 0x06 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x28 0x7E` (Set IO6 as output)

Response: `<Start> + <XTS_SPR_ACK> + <CRC> + <End>`

Protocol codes:

Name	Value
XTS_SPC_IOPIN	0x40
XTS_SPCIOP_SETCONTROL	0x10
XTS_SPR_ACK	0x10

[Pin_id(i)]:

Name	Value
IO1 - IO6	0x00000001 - 0x00000006

[Pin_setup(i)]:

Value	Comments
XTID_IOPIN_SETUP_INPUT	0x00000000
XTID_IOPIN_SETUP_OUTPUT	0x00000001

[Pin_feature(i)]:

Value	Comments
0x00000000	Disable all IO pin features.
0x00000001	Configure according to datasheet default.
0x00000002	Passive - Set and get IO pin level from host.

5.2.11 Set iopin_value

Send: `<Start> + <XTS_SPC_IOPIN> + <XTS_SPCIOP_SETVALUE> + [Pin_id(i)] + [Pin_value(i)] + <CRC> + <End>`

Example:

`0x7D 0x40 0x20 0x06 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x1A 0x7E` (Set IO6 to Hight)

`0x7D 0x40 0x20 0x06 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x1B 0x7E` (Set IO6 to Low)

Response: `<Start> + <XTS_SPR_ACK> + <CRC> + <End>`

Protocol codes:

Name	Value
XTS_SPC_IOPIN	0x40
XTS_SPCIOP_SETVALUE	0x20
XTS_SPR_ACK	0x10

[Pin_value(i)] :

Value	Comments
0x00000000	Low
0x00000001	High

5.3 Commands for XeThru Sensor Profile Configuration

The X4 XeThru sensors provide data output for applications and have a command set for application profile configuration. Users can switch between run mode and stop mode to enable or disable sensor data message output.

5.3.1 Set debug_level

Sets debug level in the XeThru module.

Send: `<Start> + <XTS_SPC_DEBUG_LEVEL> + <Debug_level> + <CRC> + <End>`

Response: `<Start> + <XTS_SPR_ACK> + <CRC> + <End>`

Protocol codes:

Name	Value
XTS_SPC_DEBUG_LEVEL	0xb0
XTS_SPR_ACK	0x10

<Debug_level>:

Name	Index	Description
None	0	No trace messages are being forwarded.
Error	1	Critical errors that is causing serious problems for the operation of the system. E.g. missing frames, memory is full.
Warning	2	Non-critical runtime notifications, indicating a problem. E.g. no RW-parameters in flash caused parameter reset.
Info	3	Providing information regarding the system status. E.g. profiling timings, memory usage etc.
Debug	4	Low level debug messages.

5.3.2 Load profile

Loads the given sensor module profile.

Send: `<Start> + <XTS_SPC_MOD_LOADAPP> + [AppID(i)] + <CRC> + <End>`

Example: `0x7D 0x21 0xAD 0x57 0x4E 0x06 0xEE 0x7E` (load XTS_ID_APP_RESPIRATION_2 profile)

Response: `<Start> + <XTS_SPR_ACK> + <CRC> + <End>`

Protocol codes:

Name	Value
XTS_SPC_MOD_LOADAPP	0x21
XTS_SPR_ACK	0x10

[AppID(i)] value is profile supported by different modules:

Module	Name	Value	Comments
X2M200	XTS_ID_APP_RESP	0x1423a2d6	Respiration
	XTS_ID_APP_SLEEP	0x00f17b17	Presence
X4M200 X4M210	XTS_ID_APP_RESPIRATION_2	0x064e57ad	Adult, 0.4-5.0m range, Feature Optimized
	XTS_ID_APP_RESPIRATION_3	0x47fabeba	Baby, 0.4-5.0m range, Feature Optimized
	XTS_ID_APP_RESPIRATION_4	0x4ac5d074	Adult, 0.4-3.0m range, Memory Optimized
	XTS_ID_APP_RESPIRATION_5	0xa9e03260	Baby, 0.4-3.5m range, Memory Optimized
X4M210	XTS_ID_APP_HEARTRATE	0x6b5c1609	heart rate, 0.4-5.0m range
X4M300	XTS_ID_APP_PRESENCE_2	0x014d4ab8	Presence

5.3.3 LED control

Use this command to choose the behavior of the sensor LED. There are three levels of LED operations, Off, Simple and Full. Different profiles may use the LED differently, but in general, the three levels OFF, SIMPLE and FULL. More information can be found at module datasheet.

Send: `<Start> + <XTS_SPC_MOD_SETLEDCONTROL> + <Mode> + <Reserved> + <CRC> + <End>`

Response: `<Start> + <XTS_SPR_ACK> + <CRC> + <End>`

Protocol codes:

Name	Value
XTS_SPC_MOD_SETLEDCONTROL	0x24
XTS_SPR_ACK	0x10

<Mode>:

Name	Value
XT_UI_LED_MODE_OFF	0x00
XT_UI_LED_MODE_SIMPLE	0x01
XT_UI_LED_MODE_FULL	0x02

5.3.4 Set detection_zone

Set the desired detection zone.

Send: `<Start> + <XTS_SPC_APPCOMMAND> + <XTS_SPCA_SET> + [XTS_ID_DETECTION_ZONE (i)] + [Start(f)] + [End(f)] + <CRC> + <End>`

Response: `<Start> + <XTS_SPR_ACK> + <CRC> + <End>`

Protocol codes:

Name	Value
XTS_SPC_APPCOMMAND	0x10
XTS_SPCA_SET	0x10
XTS_ID_DETECTION_ZONE	0x96a10a1c
XTS_SPR_ACK	0x10

5.3.5 Set sensitivity

Set the given sensitivity level.

Send: `<Start> + <XTS_SPC_APPCOMMAND> + <XTS_SPCA_SET> + [XTS_ID_SENSITIVITY(i)] + [Sensitivity(i)] + <CRC> + <End>`

Response: `<Start> + <XTS_SPR_ACK> + <CRC> + <End>`

Sensitivity value must be between 0 (low sensitivity) and 9 (high sensitivity).

Protocol codes:

Name	Value
XTS_SPC_APPCOMMAND	0x10
XTS_SPCA_SET	0x10
XTS_ID_SENSITIVITY	0x10a5112b
XTS_SPR_ACK	0x10

5.3.6 Set noisemap_control

Configure the use of noise map.

Send: `<Start> + <XTS_SPC_MOD_NOISEMAP> + <XTS_SPCN_SETCONTROL> + [Control(i)] + <CRC> + <End>`

Example:

`0x7D 0x25 0x10 0x03 0x00 0x00 0x00 0x4B 0x7E` (Noise Map Settings 3, Default setting, use stored noise map as priority, otherwise initialize noise map at reset)

`0x7D 0x25 0x10 0x06 0x00 0x00 0x00 0x4E 0x7E` (Noise Map Settings 6, Use default noise map)

Response: `<Start> + <XTS_SPR_ACK> + <CRC> + <End>`

Protocol codes:

Name	Value
XTS_SPC_MOD_NOISEMAP	0x25
XTS_SPCN_SETCONTROL	0x10
XTS_SPR_ACK	0x10

[Control(i)] values are Noise Map Settings, see below table.

Use stored Noise Map	Adapt Noise Map	Use default Noise Map	Value	Feature Description		
				On Startup	Fast startup (**)	Runtime
On	Off	On	0	If a valid Stored Noise Map exists, Active Noise Map is preloaded with Stored Noise Map. Otherwise Active Noise Map is preloaded with Default Noise Map.	Yes	Noise Map will not change.
On	Off	Off	1	If a valid Stored Noise Map exists, Active Noise Map is preloaded with Stored Noise Map. Otherwise a Noise Map is created and used both as the Active Noise Map and saved as Stored Noise Map.	Yes if stored Noise Map found in flash, otherwise no.	Noise Map will not change.
On	On	On	2	If a valid Stored Noise Map exists, Active Noise Map is preloaded with Stored Noise Map. Otherwise Active Noise Map is preloaded with Default Noise Map.	Yes	Noise Map will continuously adapt. (***)
On	On	Off	3(*)	If a valid Stored Noise Map exists, Active Noise Map is preloaded with Stored Noise Map. Otherwise a Noise Map is created and used both as the Active Noise Map and saved as Stored Noise Map.	Yes if stored Noise Map found in flash, otherwise no.	Noise Map will continuously adapt. (***)
Off	Off	On	4		Yes	

Use stored Noise Map	Adapt Noise Map	Use default Noise Map	Value	Feature Description		
				On Startup	Fast startup (**)	Runtime
				Active Noise Map is preloaded with Default Noise Map.		Noise Map will not change.
Off	Off	Off	5	A Noise Map is created and used both as the Active Noise Map. Stored Noise Map does not change.	No	Noise Map will not change.
Off	On	On	6	Active Noise Map is preloaded with Default Noise Map.	Yes	Noise Map will continuously adapt. (***)
Off	On	Off	7	A Noise Map is created and used both as the Active Noise Map. Stored Noise Map does not change.	No	Noise Map will continuously adapt. (***)

Note:

(*): Default setting

(**): Fast startup: No noise map initialization time added.

(***): Supports periodic noise map update.

Faded options are not available, which means Adapt Noise Map is always enabled.

5.3.7 Store noise map

Store the current noise map to module flash.

Fails if a store already is active, for example during the first initialize with XTID_NOISEMAP_CONTROL_INIT_ON_RESET disabled.

Send: `<Start> + <XTS_SPC_APPCOMMAND> + <XTS_SPCA_STORE_NOISEMAP> + <CRC> + <End>`

Response: `<Start> + <XTS_SPR_ACK> + <CRC> + <End>`

Protocol codes:

Name	Value
XTS_SPC_APPCOMMAND	0x10
XTS_SPCA_STORE_NOISEMAP	0x13

Name	Value
XTS_SPR_ACK	0x10

5.3.8 Load noise map

Load a previously stored noise map.

Send: `<Start> + <XTS_SPC_APPCOMMAND> + <XTS_SPCA_LOAD_NOISEMAP> + <CRC> + <End>`

Response: `<Start> + <XTS_SPR_ACK> + <CRC> + <End>`

Protocol codes:

Name	Value
XTS_SPC_APPCOMMAND	0x10
XTS_SPCA_LOAD_NOISEMAP	0x14
XTS_SPR_ACK	0x10

5.3.9 Delete noise map

Delete a previously stored noise map from flash.

Send: `<Start> + <XTS_SPC_APPCOMMAND> + <XTS_SPCA_DELETE_NOISEMAP> + <CRC> + <End>`

Response: `<Start> + <XTS_SPR_ACK> + <CRC> + <End>`

Protocol codes:

Name	Value
XTS_SPC_APPCOMMAND	0x10
XTS_SPCA_LOAD_NOISEMAP	0x15
XTS_SPR_ACK	0x10

5.3.10 Set output_control

After the profile is loaded, the profile may be able to output different kinds of messages. This command is used to enable or disable message output.

Send: `<Start> + <XTS_SPC_OUTPUT> + <XTS_SPCO_SETCONTROL> + [Output_feature(i)] + [Output_control(i)] + <CRC> + <End>`

Example: `0x7D 0x41 0x10 0x0d 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x20 0x7E` (Enable XTS_ID_BASEBAND_AMPLITUDE_PHASE as output)

Response: `<Start> + <XTS_SPR_ACK> + <CRC> + <End>`

Protocol codes:

Name	Value
XTS_SPC_OUTPUT	0x41
XTS_SPCO_SETCONTROL	0x10
XTS_SPR_ACK	0x10

[Output_feature(i)] value is the message under specified profile:

Message	Codes	Profile	Module	Comments
XTS_ID_RESP_STATUS	0x2375fe26	XTS_ID_APP _RESPIRATION_2, XTS_ID_APP _RESPIRATION_3, XTS_ID_APP _RESPIRATION_4, XTS_ID_APP _RESPIRATION_5,	X4M200 X4M210	Settings are kept on reset
XTS_ID_SLEEP_STATUS	0x2375a16c			
XTS_ID_VITAL_SIGNS	0x20020102			
XTS_ID_RESPIRATION _MOVINGLIST	0x610a3b00			
XTS_ID_RESPIRATION _NORMALIZEDMOVEMENTLIST	0xC3A331CF			
XTS_ID_RESPIRATION _DETECTIONLIST	0x610a3b02			
XTS_ID_VITAL_SIGNS	0x20020102			
XTS_ID_PRESENCE_SINGLE	0x723bfa1e	XTS_ID_APP _PRESENCE_2	X4M300	
XTS_ID_PRESENCE_MOVINGLIST	0x723bfa1f			
XTS_ID_BASEBAND_IQ	0x0000000c	ALL	X4M200 X4M300	Disabled after reset
XTS_ID_BASEBAND _AMPLITUDE_PHASE	0x0000000d			
XTS_ID_PULSEDOPPLER_FLOAT	0x00000010			
XTS_ID_PULSEDOPPLER_BYTE	0x00000011			
XTS_ID_NOISEMAP_FLOAT	0x00000012			
XTS_ID_NOISEMAP_BYTE	0x00000013			

[Output_control(i)] value decide if the message output is enabled or not.

Name	Value
XTID_OUTPUT_CONTROL_ENABLE	0x00000001
XTID_OUTPUT_CONTROL_DISABLE	0x00000000

For pulse-Doppler and noise map byte/float, bits in [Output_control(i)] decide:

Position	Name
Bit 0	XTID_OUTPUT_CONTROL_PD_ENABLE
Bit 1	XTID_OUTPUT_CONTROL_PD_SLOW_ENABLE
Bit 2	XTID_OUTPUT_CONTROL_PD_FAST_ENABLE

Note: Data throughput limitations may prevent output of multiple messages at the same time.

6 Radar Data Messages from XeThru Modules

This chapter describes the structure of raw radar messages from X4 XeThru Modules.

6.1 Radar Raw Data Message Output

Radar raw data messages are output as generic float data.

6.1.1 Generic Float Data Message

Message: [XTS_FLAGSEQUENCE_START_NOESCAPE(i)] + [PacketLength(i)] + <RESERVED> + <XTS_SPR_DATA> + <XTS_SPRD_FLOAT> + [ContentId(i)] + [Info(i)] + [Length(i)] + [DataItems(f)]

This message contains generic float data in binary format.

Output:

- ContentId: Generic Content ID, depending on application.
- Info: Generic Info. E.g. frame counter for XEP.
- Length: Number of float values in data set.
- Data: Array of float values. Depending on downconversion is enabled or not, the message data is baseband IQ data (first half I, second half Q) or RF data.

Protocol codes:

Name	Value
XTS_SPR_DATA	0xA0
XTS_SPRD_FLOAT	0x12

6.2 Application Data Message Output from XeThru Sensor

6.2.1 Radar Baseband Data Messages

X4M200 and X4M300 supports radar baseband data messages output under all profiles.

Baseband IQ Message

Outputs radar baseband IQ data.

Message: [XTS_FLAGSEQUENCE_START_NOESCAPE(i)] + [PacketLength(i)] + <RESERVED> + <XTS_SPR_APPDATA> + [XTS_ID_BASEBAND_IQ(i)] + [Counter(i)] + [NumOfBins(i)] + [BinLength(f)] + [SamplingFrequency(f)] + [CarrierFrequency(f)] + [RangeOffset(f)] + [SigIItems(f)] + ... + [SigQItems(f)]

Output:

- Counter: A sequential counter from the radar data. Incremented for each data message. Data output rate is the frame rate.
- NumOfBins: Number of bins in data set.
- BinLength: Length in meters between each bin.
- SamplingFrequency: Chip sampling frequency in Hz.
- CarrierFrequency: Chip carrier frequency in Hz.
- RangeOffset: Start of the first range bin in meters.
- SigIItems: Array of NumOfBins float values of the signal I-channel.
- SigQItems: Array of NumOfBins float values of the signal Q-channel.

Protocol codes:

Name	Value
XTS_SPR_APPDATA	0x50
XTS_ID_BASEBAND_IQ	0x0000000c

Baseband AP Message

Outputs radar baseband amplitude-phase data.

Message: [XTS_FLAGSEQUENCE_START_NOESCAPE(i)] + [PacketLength(i)] + <RESERVED> + <XTS_SPR_APPDATA> + [XTS_ID_BASEBAND_AP(i)] + [Counter(i)] + [NumOfBins(i)] + [BinLength(f)] + [SamplingFrequency(f)] + [CarrierFrequency(f)] + [RangeOffset(f)] + [PowerItems(f)] + ... + [PhaseItems(f)]

Output:

- Counter: A sequential counter from the radar data. Incremented for each data message. Data output rate is the frame rate.
- NumOfBins: Number of bins in data set.
- BinLength: Length in meters between each bin.
- SamplingFrequency: Chip sampling frequency in Hz.
- CarrierFrequency: Chip carrier frequency in Hz.
- RangeOffset: Start of the first range bin in meters.
- PowerItems: Array of NumOfBins float values of the signal power.
- PhaseItems: Array of NumOfBins float values of the signal phase.

Protocol codes:

Name	Value
XTS_SPR_APPDATA	0x50
XTS_ID_BASEBAND_AP	0x0000000d

PulseDoppler/NoiseMap Message

Outputs radar PulseDoppler/NoiseMap message.

Message: [XTS_FLAGSEQUENCE_START_NOESCAPE(i)] + [PacketLength(i)] + <RESERVED> + <XTS_SPR_APPDATA> + [DataType(i)] + [Counter(i)] + [MatrixCounter(i)] + [RangeIdx(i)] + [RangeBins(i)] + [FrequencyCount(i)] + [PulseDopplerInstance(i)] + [FPS(f)] + [FPSDecimated(f)] + + [FrequencyStart(f)] + [FrequencyStep(f)] + [Range(f)] + [Data_float/Data_byte]

Output:

- Counter: A sequential counter from the radar data. Incremented for each data message. Data output rate is the frame rate.
- MatrixCounter: Incremental matrix counter.
- RangeIdx: Range bin index of current doppler vector [0..RangeBins-1].
- RangeBins: Number of total range bins in the range-Doppler output matrix.
- FrequencyCount: Number of points in frequency axis.
- PulseDopplerInstance: Selected pulse-Doppler type from [0..N-1] where N is number of PDs.
- ByteStepStart: Start of dB compression range.
- ByteStepSize: Size of one step in dB.
- FPS: Output chip framerate [frames per second].
- FPSDecimated: Input FPS of this PulseDopplerInstance.
- FrequencyStart: Frequency of first value.
- FrequencyStep: Difference between each frequency bin.
- Range: Absolute range of current frequency array.
- Data_float: Float array, power of pulse-Doppler bin.
- Data_byte: Unsigned integer (8bits) array, power of pulse-Doppler bin (compressed float values). Decompressed float value is calculated using:
  float = powf (10.0f, (ByteStepStart + byte * ByteStepSize) / 10.0f)

Protocol codes:

Name	Value
XTS_SPR_APPDATA	0x50

[DataType(i)]:

Name	Value
XTS_ID_PULSEDOPPLER_FLOAT	0x00000010
XTS_ID_PULSEDOPPLER_BYTE	0x00000011
XTS_ID_NOISEMAP_FLOAT	0x00000012
XTS_ID_NOISEMAP_BYTE	0x00000013

6.2.2 X4M200/X4M210 Profile Messages

X4M200/X4M210 supports adult respiration profiles and baby respiration profiles.

Profile Name	Value	Comments
XTS_ID_APP_RESPIRATION_2	0x064e57ad	Adult, 0.4-5.0m range, Feature Optimized
XTS_ID_APP_RESPIRATION_3	0x47fabeba	Baby, 0.4-5.0m range, Feature Optimized
XTS_ID_APP_RESPIRATION_4	0x4ac5d074	Adult, 0.4-3.0m range, Memory Optimized
XTS_ID_APP_RESPIRATION_5	0xa9e03260	Baby, 0.4-3.5m range, Memory Optimized

X4M210 also adds support to heart rate profile:

Profile Name	Value	Comments
XTS_ID_APP_HEARTRATE	0x6b5c1609	heart rate, 0.4-5.0m range

All of them can output Respiration Legacy message, Respiration Sleep message, Respiration Moving List message, Respiration Normalized Movement List message, Respiration Detectionlist message.

Respiration profile support following [StateCode(i)]:

StateCode	Value	Description
XTS_VAL_RESP_STATE_BREATHING	0	Valid RPM detected
XTS_VAL_RESP_STATE_MOVEMENT	1	Detects motion, but cannot identify breath
XTS_VAL_RESP_STATE_MOVEMENT_TRACKING	2	Detects motion, possible breathing
XTS_VAL_RESP_STATE_NO_MOVEMENT	3	No movement detected
XTS_VAL_RESP_STATE_INITIALIZING	4	Initializing sensor
XTS_VAL_RESP_STATE_ERROR	5	Sensor has detected some problem. StatusValue indicates problem.
XTS_VAL_RESP_STATE_UNKNOWN	6	Undefined state.
XTS_VAL_RESP_STATE_HEART_RATE_AND_BREATHING	7	Heart rate and breathing

Respiration Message

Legacy message, compatible with X2M200 RESP message.

Outputs the legacy RESP message of the respiration profiles, with data when available.

Message: `<Start> + <XTS_SPR_APPDATA> + [XTS_ID_RESP_STATUS(i)] + [Counter(i)] + [StateCode(i)] + [StateData(i)] + [Distance(f)] + [BreathingPattern(f)] + [SignalQuality(i)] + <CRC> + <End>`

Output:

Parameter	Type	Description
Counter	uint32_t	Frame counter for sensor.
StateCode	uint32_t	Profile state.
StateData	uint32_t	RPM, respirations per minute (only BREATHING and HEART_RATE_AND_BREATHING states available).
Distance	float	Distance to where respiration is detected (only BREATHING and HEART_RATE_AND_BREATHING states available).
BreathingPattern	float	Relative movement of the respiration, in mm (only BREATHING and HEART_RATE_AND_BREATHING states available).
SignalQuality	uint32_t	A measure of the signal quality. Typically used to identify if the sensor is positioned correctly. Value from 0 to 10 where 0=low and 10=high (only BREATHING and HEART_RATE_AND_BREATHING states available).

Protocol codes:

Name	Value
XTS_SPR_APPDATA	0x50
XTS_ID_RESP_STATUS	0x2375fe26

Sleep Message

Compatible with X2M200 SLEEP message.

Outputs the Sleep message of the respiration profiles, with data when available.

Message: `<Start> + <XTS_SPR_APPDATA> + [XTS_ID_SLEEP_STATUS(i)] + [Counter(i)] + [StateCode(i)] + [RespirationsPerMinute(f)] + [Distance(f)] [SignalQuality(i)] + [MovementSlow(f)] + [MovementFast(f)] <CRC> + <End>`

Output:

Parameter	Type	Description
Counter	uint32_t	Frame counter for sensor.
StateCode	uint32_t	Profile state.
RespirationsPerMinute	float	RPM, respirations per minute (only BREATHING and HEART_RATE_AND_BREATHING states available).
Distance	float	Distance to where respiration is detected (only BREATHING and HEART_RATE_AND_BREATHING states available).

Parameter	Type	Description
SignalQuality	uint32_t	A measure of the signal quality. Typically used to identify if the sensor is positioned correctly. Value from 0 to 10 where 0=low and 10=high. (only BREATHING and HEART_RATE_AND_BREATHING states available).
MovementSlow	float	A measure of movement with long integration time. It is the average value of MovementSlow items.
MovementFast	float	A measure of movement with short integration time. It is the average value of MovementFast items.

Protocol codes:

Name	Value
XTS_SPR_APPDATA	0x50
XTS_ID_SLEEP_STATUS	0x2375a16c

Respiration Movement List Message

Outputs the Respiration Movement List message of the respiration profiles, with data regarding moving targets when available.

Message: `<Start> + <XTS_SPR_APPDATA> + [XTS_ID_RESPIRATION_MOVINGLIST(i)] + [Counter(i)] + [MovementIntervalCount(i)] + [MovementSlowItems(f)] + ... + [MovementFastItems(f)] + ... + <CRC> + <End>`

Output:

Parameter	Type	Description
Counter	uint32_t	Frame counter as basis for sensor.
MovementIntervalCount	uint32_t	MovementIntervalCount is the number of items in the MovementSlowItems and MovementFastItems list. Deterministic, depending on DetectionZone.
MovementSlowItems	float array	List of movement slow values, first item is average value of all MovementSlowItems.
MovementFastItems	float array	List of movement fast values, first item is average value of all MovementFastItems.

Protocol codes:

Name	Value
XTS_SPR_APPDATA	0x50
XTS_ID_RESPIRATION_MOVINGLIST	0x610a3b00

Respiration Normalized Movement List Message

Outputs the Respiration Normalized Movement List message of the respiration profiles, with data regarding moving targets when available.

Message: `<Start> + <XTS_SPR_APPDATA> + [XTS_ID_RESPIRATION_NORMALIZEDMOVEMENTLIST (i)] + [Counter(i)] + [Start(f)] + [BinLength(f)] + [Count(i)] + [NormalizedMovementSlowItems(f)] + ... + [NormalizedMovementFastItems(f)] + ... + <CRC> + <End>`

Output:

Parameter	Type	Description
Counter	uint32_t	Frame counter as basis for sensor.
Start	float	Start distance.
BinLength	float	Length in meters between each bin.
Count	uint32_t	Count is the number of items in the NormalizedMovementSlowItems and NormalizedMovementFastItems list. Deterministic, depending on DetectionZone.
NormalizedMovementSlowItems	float array	List of normalized movement slow values.
NormalizedMovementFastItems	float array	List of normalized movement fast values.

Protocol codes:

Name	Value
XTS_SPR_APPDATA	0x50
XTS_ID_RESPIRATION_NORMALIZEDMOVEMENTLIST	0xC3A331CF

Respiration Detection List Message

Outputs the Respiration Detection List message of the respiration profiles.

Message: `<Start> + <XTS_SPR_APPDATA> + [XTS_ID_RESPIRATION_DETECTIONLIST(i)] + [Counter(i)] + [DetectionCount(i)] + [DetectionDistanceItems(f)] + ... + [DetectionRadarCrossSectionItems(f)] + ... + [DetectionVelocityItems(f)] + ... + <CRC> + <End>`

Output:

Parameter	Type	Description
Counter	uint32_t	Frame counter for sensor.
DetectionCount	uint32_t	

Parameter	Type	Description
		Number of detections observed, listed in DetectionDistanceItems, DetectionRadarCrossSectionItems and DetectionVelocityItems lists.
DetectionDistanceItems	float array	List of distance to detections in meters. The DetectionDistanceItems only provides up two items. At Movement state, it will only output the distance of the closest object. At Breathing state, it outputs two items: [Closest target distance, Breathing target distance]. At other states, the DetectionDistanceItems will be empty.
DetectionRadarCrossSectionItems	float array	List of radar cross sections of detections, unit is dB(m^2).
DetectionVelocityItems	float array	List of radial velocity of detected objects in meters/second. (not implemented)

Protocol codes:

Name	Value
XTS_SPR_APPDATA	0x50
XTS_ID_RESPIRATION_DETECTIONLIST	0x610a3b02

Vital Signs Message

Outputs the Respiration Detection List message of the respiration profiles.

Message: `<Start> + <XTS_SPR_APPDATA> + [XTS_ID_VITAL_SIGNS(i)] + [Counter(i)] + [StateCode(i)] + [RespirationsRate(f)] + [RespirationsDistance(f)] + [RespirationsConfidence(f)] + [HeartRate(f)] + [HeartDistance(f)] + [HeartConfidence(f)] + [NormalizedMovementSlow(f)] + [NormalizedMovementFast(f)] + [NormalizedMovementStart(f)] + [NormalizedMovementEnd(f)] + <CRC> + <End>`

Output:

Parameter	Type	Description
Counter	uint32_t	Frame counter for sensor.
StateCode	uint32_t	Profile state.
RespirationsRate	float	Respirations per minute (only BREATHING and HEART_RATE_AND_BREATHING states available).
RespirationsDistance	float	Distance to where respiration is detected (only BREATHING and HEART_RATE_AND_BREATHING states available).
RespirationsConfidence	float	Respiration rate confidence level.

Parameter	Type	Description
HeartRate	float	Heart per minute (only X4M210 XTS_ID_APP_HEARTRATE profie HEART_RATE_AND_BREATHING state available).
HeartDistance	float	Distance to where heart is detected (only X4M210 XTS_ID_APP_HEARTRATE profie HEART_RATE_AND_BREATHING state available).
HeartConfidence	float	Hear rate confidence level. (only X4M210 XTS_ID_APP_HEARTRATE profie HEART_RATE_AND_BREATHING state available).
NormalizedMovementSlow	float	The sum of values in the normalized slow movement list array in 1meter with target in middle.
NormalizedMovementFast	float	The sum of values in the normalized fast movement list array in 1meter with target in middle.
NormalizedMovementStart	float	The start point distance of the above 1 meter array.
NormalizedMovementEnd	float	The end point of the above 1 meter array.

Protocol codes:

Name	Value
XTS_SPR_APPDATA	0x50
XTS_ID_VITAL_SIGNS	0x20020102

6.2.3 X4M300 Profile Messages

X4M300 supports Presence2 profile.

Profile Name	Value
XTS_ID_APP_PRESENCE_2	0x014d4ab8

X4M300 Presence2 profile supports output of Presence Single message, Presence Moving List message.

Presence Single message

Outputs the Presence Single message of the Presence2 profile, with data when available.

Message: `<Start> + <XTS_SPR_APPDATA> + [XTS_ID_PRESENCE_SINGLE(i)] + [Counter(i)] + [PresenceState(i)] + [Distance(f)] + <Direction> + [SignalQuality(i)] + <CRC> + <End>`

Output:

Parameter	Type	Description
Counter	uint32_t	Frame counter for sensor.

Parameter	Type	Description
PresenceState	uint32_t	Sensor state output. See StateCode values table.
Distance	float	Distance to where presence is detected.
Direction	uint8_t	Direction of detected object. 0=stationary, 1=towards sensor, 2=away from sensor.
SignalQuality	uint32_t	A measure of the signal quality. Typically used to identify if the sensor is positioned correctly. Value from 0 to 10 where 0=low and 10=high. (Presence state only).

[PresenceState(i)]:

PresenceState	Value	Description
XTS_VAL_PRESENCE_PRESENCESTATE_NO_PRESENCE	0	No presence detected
XTS_VAL_PRESENCE_PRESENCESTATE_PRESENCE	1	Presence detected
XTS_VAL_PRESENCE_PRESENCESTATE_INITIALIZING	2	The sensor initializes after the True Presence Profile is executed
XTS_VAL_PRESENCE_PRESENCESTATE_UNKNOWN	3	The sensor is in an unknown state and requires a Profile and User Settings to be loaded.

Protocol codes:

Name	Value
XTS_SPR_APPDATA	0x50
XTS_ID_PRESENCE_SINGLE	0x723bfa1e

Presence Moving List Message

Outputs the Presence Moving List message of the Presence2 profile, with data regarding moving targets when available.

Message: `<Start> + <XTS_SPR_APPDATA> + [XTS_ID_PRESENCE_MOVINGLIST(i)] + [Counter(i)] + [PresenceState(i)] + [MovementIntervalCount(i)] + [DetectionCount(i)] + [MovementSlowItems(f)] + ... + [MovementFastItems(f)] + ... + [DetectionDistanceItems(f)] + ... + [DetectionRadarCrossSectionItems(f)] + ... + [DetectionVelocityItems(f)] + ... + <CRC> + <End>`

Output:

Parameter	Type	Description
Counter	uint32_t	Frame counter for sensor.
PresenceState	uint32_t	Sensor state output. See StateCode values table.

Parameter	Type	Description
MovementIntervalCount	uint32_t	MovementIntervalCount is the number of items in the MovementSlowItems and MovementFastItems list. Its value depends on DetectionZone.
DetectionCount	uint32_t	Number of detections observed, listed in DetectionDistanceItems, DetectionRadarCrossSectionItems and DetectionVelocityItems lists.
MovementSlowItems	float array	List of movement slow values, first item is average value of all MovementSlowItems.
MovementFastItems	float array	List of movement fast values, first item is average value of all MovementFastItems.
DetectionDistanceItems	float array	List of distance to detections in meters. The DetectionDistanceItems only provides up two items. At Movement state, it will only output the distance of the closest object. At Breathing state, it outputs two items: [Closest target distance, Breathing target distance]. At other states, the DetectionDistanceItems will be empty. (not implemented)
DetectionRadarCross SectionItems	float array	List of radar cross sections of detections, unit is dB(m^2). (not implemented)
DetectionVelocityItems	float array	List of radial velocity of detected objects in meters /second. (not implemented)

[PresenceState(i)]:

PresenceState	Value	Description
XTS_VAL_PRESENCE_PRESENCESTATE_NO_PRESENCE	0	No presence detected
XTS_VAL_PRESENCE_PRESENCESTATE_PRESENCE	1	Presence detected
XTS_VAL_PRESENCE_PRESENCESTATE_INITIALIZING	2	The sensor initializes after the True Presence Profile is executed
XTS_VAL_PRESENCE_PRESENCESTATE_UNKNOWN	3	The sensor is in an unknown state and requires a Profile and User Settings to be loaded

Protocol codes:

Name	Value
XTS_SPR_APPDATA	0x50
XTS_ID_PRESENCE_MOVINGLIST	0x723bfa1f

7 References

[1]	Novelda, "XEP source ", https://www.xethru.com/community/resources/xep-source.90/
[2]	Novelda, "Module Connector", https://www.xethru.com/community/resources/module-connector-windows.78/
[3]	Novelda, "Module Communication Protocol Wrapper", https://www.xethru.com/community/resources/module-communication-protocol-wrapper.97/
[4]	Novelda, "XeThru Serial Protocol", https://www.xethru.com/community/resources/xethru-serial-protocol.14/
[5]	Novelda, "X4 Datasheet", https://www.xethru.com/community/resources/x4-datasheet.106/

8 Document History

Rev.	Release date	XEP version	Change description
A	2018-May-15	3.0.0	Initial release
B	2018-August-09	3.4.7	1. Fix errors 2. Update noise map commands 3. Respiration profile add normalized movement message
C	2018-November-6	4.1.0	1. Update noise map commands description 2. add X4M210 content

9 Disclaimer

The information in this document is provided in connection with Novelda products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Novelda products. EXCEPT AS SET FORTH IN THE NOVELDA TERMS AND CONDITIONS OF SALES LOCATED ON THE NOVELDA WEBSITE, NOVELDA ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL NOVELDA BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS AND PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF NOVELDA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Novelda makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and products descriptions at any time without notice. Novelda does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Novelda products are not suitable for, and shall not be used in, automotive applications. Novelda products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.