Software Modem: Reference Manual

Contents

Table of Figures

Table of Tables

1 Introduction and Objective

The Software Modem (referred to modem) is a software set running on a LoRa based module to provide an API for end node operation.

The Software Modem includes the LoRaWAN MAC layer functions, and some application layer functions provided in Modem Services when using Semtech Cloud Service.

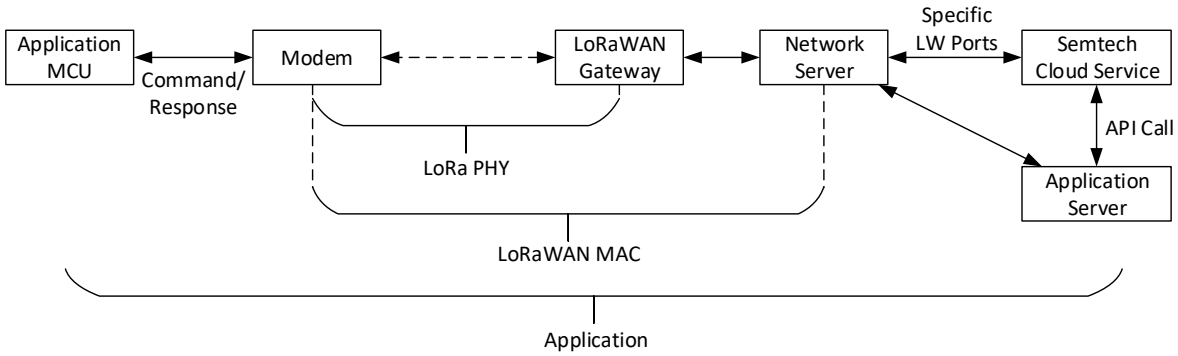

Figure 1 Scope of the modem and modem service

The physical connection between the application MCU (also called host MCU) and the modem: is described in chapter 2. The message flow timing is described in chapter 4. The definition of command and response is described in chapters 5, 6 and 7. Examples are available in chapter 8.

The cloud service related reference is in Modem Service and Reliable Octet Stream Encoding.

2 Modem Pinout

Pin No.	Terminal Name	Type	Description
1	PA12	O	TCXO power supply output. Must be externally connected to pin 48.
2	PA11	I/O	No Connect
3	GND	GND	Ground.
4	VDD_PA12	PWR	Power supply for TCXO (pin 1).
5	VDD_MCU	PWR	Power supply from modem MCU.
6	VDD_RF	PWR	Power supply for RF section.
7	GND	GND	Ground.
8	DBG_SX1276_DIO2	—	Internal use. Must be left n/c.
9	DBG_SX1276_DIO3	—	Internal use. Must be left n/c.
10	DBG_SX1276_DIO4	—	Internal use. Must be left n/c.
11	DBG_SX1276_DIO5	—	Internal use. Must be left n/c.
12	DBG_SX1276_DIO1	—	Internal use. Must be left n/c.
13	DBG_SX1276_DIO0	—	Internal use. Must be left n/c.
14	PB15	I/O	No Connect
15	PB14	I/O	No Connect
16	PB13	I/O	No Connect
17	PB12	I/O	No Connect
18	PA10/USART1_RX	I	UART RX (Serial port: host-to-modem)
19	PA9/USART1_TX	O	UART TX (Serial port: modem-to-host)
20	PA8	O	BUSY signaling line
21	PA5	O	Diagnostics LED (optional). Active high, connect to LED with appropriate in-series resistor.
22	PA4	I/O	No Connect
23	PA3	I	Debug UART RX. No connect, recommend routing to test point.
24	PA2	O	Debug UART TX. No connect, recommend routing to test point.
25	GND	GND	Ground.
26	ANT	ANA	Transmit / Receive Antenna connection.
27	GND	GND	Ground.
28	DBG_CRF1	—	Internal use. Must be left n/c.
29	DBG_CRF3	—	Internal use. Must be left n/c.
30	DBG_CRF2	—	Internal use. Must be left n/c.
31	NC	I	Not used. Should be left n/c.
32	VREF+	PWR	Reference voltage for modem MCU ADC section. Must be externally connected to VD
33	PA0/WKUP1	I/O	No Connect
34	MCU_nRST	I	Modem MCU reset pin. Has internal pull-up and may be left floating.
35	PB8	I	COMMAND signaling line
36	PB9	O	EVENT signaling line
37	PB2	O	MCURESET signaling line. Resets the host MCU. Active low, Z (high impedance) when inactive.
38	PB7	I/O	No Connect
39	PB6	I/O	No Connect
40	PB5	I/O	No Connect

41	PA13	I/O	Internal Use. Programming SWDIO.
42	PA14	I/O	Internal Use. Programming SWCLK.
43	BOOT0	I	No Connect. Internal use
44	GND	Ground	Ground
45	PH1-OSC_OUT	I/O	No Connect
46	PH0-OSC_IN	I/O	No Connect
47	TCXO_OUT	O	TCXO output (Optional)
48	TCXO_EN	Power	TCXO power supply input. Externally connect to pin 1.
49-57	GND	GND	Ground

3 Modem Interface

Several connections need to be established.

3.1 Serial port

- USART1
  - TX PIN
  - RX PIN

  The following serial port parameters are used.

Table 1 Serial port parameters

Baud rate	115,200 bps
Data bits	8
Stop bits	1
Parity	None
Flow control	None

3.2 GPIO Signalling Lines

3.2.1 COMMAND line

- Active-low, GPIO input line
- This line is used to signal to the modem that the host is about to transmit a command. It must be driven low at least 10 ms before sending the first character on the to allow the modem to wake up and start reception. After the command is sent, the line must be driven high, at which point the modem will process the command. If a valid command was received, the modem will transmit its response within 50 ms.

3.2.2 BUSY line

- Active-high, GPIO output line
- This line signals if the modem is busy or ready to receive commands. It is high while the modem is busy and will go low as soon as the modem is ready to receive a command. The modem should be ready to receive after a maximum of 10 ms after the COMMAND line has been asserted. The BUSY line will go high again after the command has been received and the COMMAND line has been released.

3.2.3 EVENT line

- Active-high, GPIO output line
- This line signals to the host that the modem has asynchronous event data pending. The host can use the GetEvent command to retrieve such data.

3.2.4 MCURESET line

- Active-low, high impedance when inactive, GPIO output line
- Resets the host MCU

4 Message Flow

Messages are exchanged via USART1 using COMMAND, BUSY and EVENT GPIO lines for synchronization. This section describes the timing for message flow between application MCU and the modem.
The COMMAND line is set by application MCU, and the EVENT, the BUSY line is set by the modem.
All serial communication consists strictly of host-initiated command-response sequences.

4.1 Message Flow for Command and Response

Flow for sending command and reading synchronous response:

1. Sending commands is initiated via operating COMMAND line: the application MCU pulls the COMMAND line low.
2. The modem drives the BUSY line low.
3. The application MCU sends the command using the USART1 RX line.
4. The COMMAND line must be released after the last character of the command has been transmitted. The modem will only start interpreting the command after the COMMAND line has been released.
5. The modem drives the BUSY line high.
6. If the checksum has been validated the modem will process the command and will send a matching response within 50 ms, otherwise no response will be sent. The spacing between characters of the response does not exceed 5 ms.

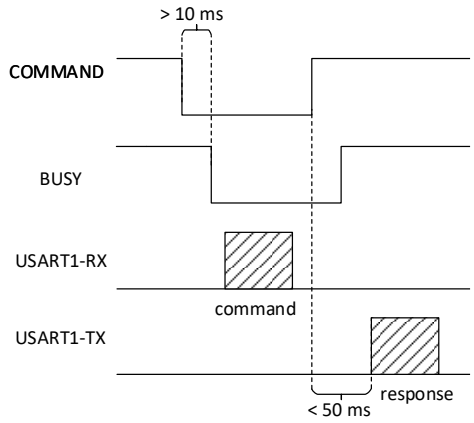

Figure 2 Timing for synchronous command/response pair

Please note that the application MCU must not send a new command before the response has been fully received.
Please note that the modem will not send any unsolicited response on the serial port. If event data becomes available asynchronously, the modem will signal this using the EVEN line.

4.2 Message Flow for Asynchronously Response

There can be an asynchronous event generated from the modem.
Flow for receiving asynchronously response:

1. Whenever data becomes available asynchronously, the modem sets the EVENT line high.
2. The application MCU should issue a GETEVENT command to retrieve the response data.
3. After all pending responses have been retrieved, the EVENT line will be set low again.

Please note that the timing requirement in section 4.1 must be fulfilled in the GETEVENT command/response pair.

5 Message Format

This section describes the packet format running on the USART1 interface, for both command packets and response packets.
The maximal packet size is 256 bytes.

5.1 Command Packets

All command messages consist of a command code which defines the operation to be performed, a payload length and optional payload, and a trailing checksum character which is the XOR of all previous bytes.

Table 2 Command message structure

Field	Size (bytes)	Description
cmd	1	Command code (see section 0)
len	1	Length of the payload field
payload	0-255	Parameters for the command as binary data
chk	1	Packet checksum (xor over cmd, len, payload)

The modem begins analyzing the command message after the COMMAND line is released. After the packet structure is successfully verified (correct number of bytes and correct check character), the modem will interpret the command and in any case, will send a response. If the packet structure is invalid, the modem ignores the packet and will not send a response.

5.2 Response Packets

The response format is similar to the command format but instead of the command code, it holds a return code which indicates the successful completion of the command or an error condition.

Table 3 Response message structure

Field	Size (bytes)	Description
rc	1	Return code of the command
len	0-255	Length of the payload field
payload	1	Response data of the command as binary data
chk	1	Packet checksum (xor over rc, len, payload)

The return code byte is defined as follows.

Table 4 List of return codes

Code	Name	Description
0x00	Ok	command executed without errors
0x01	Unknown	command code unknown
0x02	NotImpl	command not yet implemented
0x03	NotInit	command not initialized
0x04	Invalid	command parameters invalid
0x05	Busy	command cannot be executed now
0x06	Fail	command execution failed
0x07	BadFmt	format check failed
0x08	BadCrc	crc check failed
0x09	BadSig	signature verification failed
0x0A	BadSize	size check failed

6 Commands and Responses

6.1 Overview of commands and responses

The serial communication with the modem is performed using strictly alternating command and response messages. Response messages convey a return code and the result of the operation triggered by the preceding command message. All multi-byte integers contained in command or response payloads are transmitted least-significant-byte-first (LSBF).

Table 5 List of commands and responses

Name	Code	Description	Input	Output
GetEvent	0x00	Retrieve pending events		type[1], count[1], eventdata[n]
GetVersion	0x01	Get bootloader + firmware version		bootversion[4], firmwareversion
Reset	0x02	Reset modem		
FactoryReset	0x03	Perform modem factory reset		
Firmware	0x04	Write firmware update	part[4+128]	
GetTime	0x05	Get GPS wall time		timestamp[4]
SetAlarm	0x06	Set alarm / wakeup timer	seconds[4]	
GetTrace	0x07	Get diagnostic crash log		backtrace log[n]
GetPin	0x08	Get device registration PIN		PIN[4]
GetChipEui	0x09	Get device chip EUI		ChipEUI[8]
GetJoinEui	0x0A	Get join EUI		JoinEUI[8]
SetJoinEui	0x0B	Set join EUI and derive keys	JoinEUI[8]	
GetDevEui	0x0C	Get device EUI		DeviceEUI[8]
SetDevEui	0x0D	Set device EUI and derive keys	DeviceEUI[8]	
SetAppKey	0x0E	Set application key	Appkey[16]	
SetNwkKey	0x0F	Set network key	NwkKey[16]	
GetInterval	0x12	Get DM reporting interval		interval[1]
SetInterval	0x13	Set DM reporting interval	interval[1]	
GetRegion	0x14	Get regulatory region		region[1]
SetRegion	0x15	Set regulatory region	region[1]	
GetProfile	0x16	Get ADR profile		type[1]
SetProfile	0x17	Set ADR profile and optional parameters	type[1]+list[16]*	
Join	0x18	Start (re-)joining the network		
RequestTx	0x19	Transmit frame unconfirmed	port[1], data[n]	
UploadInit	0x1A	Set file upload port, encryption mode, size	p[1],en[1],sz[2]	
UploadData	0x1B	Write data for file upload transmission	data[n]	
StreamInit	0x1C	Set data stream parameters	param[2-5]	
StreamInit	**0x1C**	**Set data stream parameters**	**param[2-5]**	

StreamStatus Query the status of the streaming FIFO on the specified port. It returns two unsigned 16 bit integer values indicating the number of bytes pending transmission and the number of bytes still free in the FIFO. ➔ See also StreamData, STreamInit and GetDmPort.	0x1D	Transmit frame immediately (smoke alarm)	port[1], data[n]	
EmergencyTx				
GetCharge	0x1E	Get accumulated charge counter		charge[4]
GetMaxPayload	0x1F	Get max payload size		size[1]
SetClass	0x20	Set LoRaWAN class A/C	class[1]	
SetMulticast	0x21	Set multicast session parameters	param[40]	
SetTxPowOff	0x22	Set TX power correction offset	offset[1]	
GetDmPort	0x23	Get DM port		port[1]
SetDmPort	0x24	Set DM port	port[1]	
DmStatus	0x25	Send DM status now	inflist[n]	
GetStatus	0x26	Get modem status		status[1]
Suspend	0x27	Suspend/Resume radio operations	suspend[1]	
SetDmInfo	0x28	Set default info for DM status	inflist[n]	
StreamData	0x29	Send data stream record	port[1]+record[n]	
GetClass	0x2A	Get the LoRaWAN device class		class[1]
ListRegions	0x2B	List the supported regulatory regions		regions[1:5]
GetTxPowOff	0x2C	Get power correction offset		offset[1]
GetDmInfo	0x2D	Get default info fields for DM status		inflist[n]
SetAppStatus	0x2E	Set application-specific status for DM	appstatus[8]	
StreamStatus	0x2F	Retrieve stream status	port[1]	pending[2]+free[2]
Test	0x32	Radio test functions	(see description)	(see description)

6.2 Command Details

All multi-byte integers contained in command or response payloads are transmitted least-significant-byte-first (LSBF).

6.2.1 GetEvent

This command can be used to retrieve pending events from the modem.
Pending events are indicated by the EVENT line. The EVENT line will be de-asserted after all events have been retrieved and no further events are available. When no event is available this command returns with empty response payload.

Table 6 Response payload format for GetEvent command

Field	Size (bytes)	Description
type	1	Event type, see Table 31
count	1	Number of missed events of this type in case of overrun
data	0-253	Event-specific data

Events which are not retrieved by the application might be overwritten by new event data of the same type. In this case, only the latest event data will be returned and the count field indicates how many events of this type have been missed. See details in section 0.

6.2.2 GetVersion

This command returns the version of the bootloader and the version of the installed firmware.

Table 7 Response payload format for GetVersion command

Field	Size (bytes)
bootversion	4
fwversion	4

6.2.3 Reset

This command performs a reset of the modem MCU. All transient state (including session information) will be lost and the modem needs to join the network again.

6.2.4 FactoryReset

This command performs a factory reset. In addition to the MCU reset all persistent settings are reset back to their factory state.

6.2.5 Firmware

This command is used to repeatedly store parts of a firmware update to be installed.
The firmware data has to be split into blocks of 128 bytes each and has to be provided in ascending order.

Table 8 Command payload format for Firmware command

Field	Size (bytes)	Description
blkno	2	Current block number (0 to blkcnt-1)
blkcnt	2	Total number of blocks
blkdata	128	Firmware section starting at byte blkno*128. If the firmware size is not a multiple of 128 then the last data block must be padded with zero bytes.

6.2.6 GetTime

Query the current GPS time.
The application layer clock synchronization protocol is used to link the device clock to the GPS time. The returned time is the seconds since GPS epoch (00:00:00, Sunday 6th of January 1980). The device is not yet synchronized to GPS time if the returned value is zero.
The accuracy of the synchronization is in the range of seconds and depends on latencies in the network infrastructure.

Table 9 Command payload format for GetTime command

Field	Size (bytes)	Description
seconds	4	seconds since GPS epoch

6.2.7 SetAlarm

This command sets an application alarm timer (in seconds).
When the timer has expired, an Alarm event is generated.

6.2.8 GetTrace

This command returns the latest crashlog saved by the current firmware.
The format of the crashlog data is proprietary and will not be disclosed.

6.2.9 GetPin

This command returns the device registration PIN (4 bytes).

6.2.10 GetChipEui

This command returns the ChipEUI.
The ChipEUI is also the default DeviceEUI. It is programmed during manufacturing and is immutable.

➔ See also GetDevEui and GetJoinEui.

6.2.11 GetJoinEui

This command returns the join EUI.

➔ See also SetJoinEui, GetDevEui, and GetJoinEui

6.2.12 SetJoinEui

This command sets the join EUI and triggers a new key derivation and automatically sets a new AppKey.

➔ See also SetDevEui, GetDevEui, and GetJoinEui

6.2.13 GetDevEui

This command returns the DeviceEUI.

➔ See also SetDevEui, GetChipEui, and GetJoinEui

6.2.14 SetDevEui

This command sets the DeviceEUI and triggers a new key derivation and automatically sets a new AppKey.

➔ See also GetDevEui, GetJoinEui and SetJoinEui.

6.2.15 SetAppKey

This command sets the LoRaWAN 1.1 AppKey.
Note that a factory reset will erase this information. The device is required to rejoin the network.

➔ See also SetNwkKey, SetDevEui, and SetJoinEui.

6.2.16 SetNwkKey

This command sets the LoRaWAN 1.0.3 DevKey or LoRaWAN 1.1 NwkKey.
Note that a factory reset will erase this information. The device is required to rejoin the network.

➔ See also SetAppKey, SetDevEui, and SetJoinEui.

6.2.17 GetInterval

This command returns the device management reporting interval.
The interval is specified in seconds, minutes, hours or days.

➔ See also SetInterval and Format of Reporting Interval.

6.2.18 SetInterval

This command sets the device management reporting interval.
The interval is specified in seconds, minutes, hours or days. A value of 0 (zero) disables device management reporting.

➔ See also GetInterval and Format of Reporting Interval.

6.2.19 GetRegion

This command returns the regulatory region.

Table 10 List of region codes

Region	Code
EU868	0x01
AS923	0x02
US915	0x03
AU915	0x04
reserved	0x05

➔ See also SetRegion and List of region codes.

6.2.20 SetRegion

This command sets the regulatory region.

➔ See also GetRegion and List of region codes.

6.2.21 GetProfile

This command returns the ADR profile type.

➔ See also SetProfile.

6.2.22 SetProfile

This command sets the ADR profile and parameters.

Table 11 List of ADR profile and parameters

Name	Code	Parameters
Network Server Controlled	0x00	
Mobile Long Range	0x01	
Mobile Low Power	0x02	
Custom	0x03	list of preferred data rates [16]

➔ See also GetProfile.

6.2.23 Join

This command starts joining or rejoining the network. During the join procedure, no further transmissions can occur.
When the network has been successfully joined, a Joined event is generated.

➔ See also RequestTx.

6.2.24 RequestTx

This command requests sending the given data on the specified port as an unconfirmed frame.

It will be sent as soon as possible taking into account the current bandwidth usage of the regulatory region. A TxDone event is generated when the frame either has been sent or couldn't be sent if the specified data exceeded the maximum possible payload size. The parameter of the TxDone event indicates if the frame was sent (0x01) or not (0x00). If a RequestTx command is issued before the TxDone event of a previous RequestTx is generated, the command will fail with Busy return code. When application downlink data has been received in RX1 or RX2, a DownData event will be generated containing the port and data received.

6.2.25 See also GetMaxPayload, StreamStatus

Query the status of the streaming FIFO on the specified port. It returns two unsigned 16 bit integer values indicating the number of bytes pending transmission and the number of bytes still free in the FIFO.

- ➔ See also StreamData, STreamInit and GetDmPort.
- ➔ EmergencyTx, and Join.

6.2.26 UploadInit

This command prepares a fragmented file upload.
It specifies the port for the subsequent upload, optional encryption mode, and the file size. The port is included as metadata in the file stream and the stream is sent on the device management port so it can be processed by the Semtech Cloud Service. When an encryption mode is used, the file data is encrypted using a 128-bit AES key derived from the AppSKey before it is further processed by the upload service. Using encryption, full privacy can be ensured even if the file data is reassembled by the Semtech DM service. In this case, Semtech has no knowledge of the contents of the file data being uploaded! This command can also be used to cancel an ongoing file upload by specifying a file size of zero for the port in use.

- ➔ See also UploadData and GetDmPort.

6.2.27 UploadData

This command can be used repeatedly to set file data to be uploaded.
The file data needs to be split into parts of maximum 255 bytes each. The data is appended to an internal buffer and will only be sent after all of the indicated bytes have been submitted. The maximum size of file data is limited to 8K bytes. After the last part of the file data has been submitted, the data will be optionally encrypted, and the transmission stream is started. Redundant fragments are continuously sent in the background until a confirmation is received from the server that it has fully decoded the file or until twice the required number of chunks have been sent and no confirmation is received. When the upload stream is stopped, an UploadDone event is generated which carries a status byte which indicates success or timeout. Running file upload is indicated by the Upload flag in the modem status.

- ➔ See also UploadInit, GetDmPort and GetStatus.

6.2.28 StreamInit

The data stream encoder is initialized with specific parameters. This command must be called once before the first call to *StreamData*. If not called before the first call to *StreamData* then a default initialization takes place with port=0 and mode=0.

- ➔ See also StreamData, StreamStatus and GetDmPort.

6.2.29 StreamStatus

Query the status of the streaming FIFO on the specified port. It returns two unsigned 16 bit integer values indicating the number of bytes pending transmission and the number of bytes still free in the FIFO.

➔ See also StreamData, STreamInit and GetDmPort.

6.2.30 EmergencyTx

This command sends the given data on the specified port as an unconfirmed frame immediately.
It has a higher priority than all other services and does not take the duty cycle or payload size restrictions into account. It can be used to signal an alarm condition (like a smoke alarm) in real time, but it should be used with caution!

➔ See also RequestTx and GetMaxPayload.

6.2.31 GetCharge

This command returns the total charge counter of the modem in mAh.
This value includes the accumulated charge since the production of the modem regardless of potential battery replacements.

6.2.32 GetMaxPayload

This command returns the maximum application payload size possible according to the LoRaWAN regional parameters for the next transmission using the current data rate while assuming no FOpts are present and that a device is not behind a repeater.

➔ See also RequestTx.

6.2.33 SetClass

This command sets the LoRaWAN device class. Currently, only class A and class C are supported.
If the command is successful, a change from class A to class C is effective after a completed TX transaction. The network server should also be informed about the class change, typically on a separate channel for LoRaWAN 1.0.3. For a change from class C to class A, the RX remains enabled until the next TX transaction. Note that the class settings are not persistent.

Table 12 Command payload format for SetClass command

Field	Size (bytes)	Description
class	1	Class value

Table 13 List of LoRaWAN device class code.

Value	LoRaWAN Class	Description
0x00	A	Open short RX windows after TX
0x01	C	RX remains on

➔ See also GetClass.

6.2.34 SetMulticast

This command configures the LoRaWAN a downlink multicast session.
For details, refer to the LoRaWAN Standard Specification and the LoRaWAN Remote Multicast Setup Specification.
Multicast groups are not persistent and are always kept until the next reset. A maximum of 2 multicast groups is allowed.
If the group address is already configured it will be kept and the session parameters will be overwritten.

Table 14 Command payload format for SetMulticast command

Field	Size (bytes)	Description
grpaddr	4	Multicast group address
nwkkeydn	16	Multicast group network key to verify the MIC of downlink traffic
appkey	16	Multicast group application key to decrypt the application payload
seqnoadn	4	The initial sequence counter for the application downlinks

6.2.35 SetTxPowOff

This command sets the board-specific correction offset for transmission power to be used.
The offset depends on the board design and antenna matching and is expressed in dB (signed integer).

6.2.36 GetDmPort

This command gets the device management port (default 200).

➔ See also SetDmPort.

6.2.37 SetDmPort

This command sets the device management port (default 200).

➔ See also GetDmPort and SetDmInfo.

6.2.38 DmStatus

This command sends the specified set of information fields in one or more DM status messages immediately.
The set is specified as a list of field codes as defined in Uplink Message Format.

➔ See also SetDmInfo and Uplink Message Format.

6.2.39 GetStatus

This command returns the modem status which may indicate one or more notification conditions.

Table 15 List of status codes

Bit	Value	Name	Description
0	0x01	*Brownout*	reset after brownout
1	0x02	*Crash*	reset after panic
2	0x04	*Mute*	device is muted
3	0x08	*Joined*	device has joined the network
4	0x10	*Suspend*	radio operations suspended (low power)
5	0x20	*Upload*	file upload in progress

6.2.40 Suspend

This command temporarily suspends or resumes the modem's radio operations.
It can be used to prevent extra power consumption by the modem in case the application MCU temporarily needs more power itself and wants to prevent exceeding limits.
Operations are suspended with parameter value 0x01 and resumed with parameter value 0x00.

6.2.41 SetDmInfo

This command sets the default info fields to be included in the periodic DM status messages.
The set is specified as a list of field codes as defined in Uplink Message Format. An empty set is valid and will effectively disable the DM status message.

➔ See also GetDmPort and Uplink Message Format.

6.2.42 StreamData

This command encodes a data record and sends one or more redundancy-encoded frames on the specified port.
The size of the record has to match the payload size specified in the StreamInit command for this port. When all fragments have been sent a StreamDone event will be generated.

➔ See also StreamInit and GetDmPort.

6.2.43 GetClass

This command gets the LoRaWAN device class.

➔ See also SetClass.

6.2.44 ListRegions

This command returns the regulatory regions supported by the modem.

➔ See also GetRegion and SetRegion.

6.2.45 GetTxPowOff

This command gets the board-specific correction offset for transmission power to be used (signed integer in dB).

➔ See also SetTxPowOff.

6.2.46 GetDmInfo

This command lists the default info fields to be included in the periodic DM status messages.

➔ See also SetDmInfo and Uplink Message Format.

6.2.47 SetAppStatus

This commands sets application-specific status information to be reported to the DM service.
Arbitrary information defined by the application can be set as status data. Once the application status has been set the appstatus info field will be included in the periodic status reports. If the application desires to send the status immediately, it can issue the DmStatus command with the appstatus tag. The application status is not stored persistently, i.e. after reset, no application status will be reported.

➔ See also DmStatus and Uplink Message Format.

6.2.48 StreamStatus

Query the status of the streaming FIFO on the specified port.
It returns two unsigned 16-bit integer values indicating the number of bytes pending transmission and the number of bytes still free in the FIFO.

Table 16 Response payload format of StreamStatus

Field	Size (bytes)
pending	2
free	2

Return codes:

- **Ok**: Return 4 bytes of information
- **NotInit**: Stream is not initialized

➔ See also StreamInit and StreamData.

6.2.49 Test

This command is used to implement test functionality for regulatory conformance and certification testing.
With the exception of the TST_START command, test commands are only available if test mode is active. Test mode can only be activated if the modem has not yet received a command that results in radio operation. Once test mode is active, all other modem commands are disabled.

Table 17 Encodings for SF, BW, CR in Test commands

Value	Spreading Factor	Bandwidth	Coding Rate
0	FSK	125 kHz	4/5
1	SF7	250 kHz	4/6
2	SF8	500 kHz	4/7
3	SF9		4/8
4	SF10		
5	SF11		
6	SF12		

6.2.49.1 TST_START

Start test mode. This command enables all other test functions.

Table 18 Command payload format for TST_START command

Size (bytes)	Field
1	0x00
8	“TESTTEST”

6.2.49.2 TST_NOP

No operation. This command may be used to terminate an ongoing continuous TX operation.

Table 19 Command payload format for TST_NOP command

Size (bytes)	Field
1	0x01

6.2.49.3 TST_TX_SINGLE

Transmit a single packet.

Table 20 Command payload format for TST_TX_SINGLE command

Size (bytes)	Field
1	0x02
4	frequency
1	tx_power
1	sf
1	bw
1	cr
1	payload_length

6.2.49.4 TST_TX_CONT

Continuously transmit packets as fast as possible.

Table 21 Command payload format for TST_TX_CONT command

Size (bytes)	Field
1	0x02
4	frequency
1	tx_power
1	sf
1	bw
1	cr

1	payload_length

6.2.49.5 TST_TX_HOP

Continuously transmit packets as fast as possible while respecting regional regulatory constraints (channel hopping, output power, dwell time, duty cycle).

Table 22 Command payload format for TST_TX_HOP command

Size (bytes)	Field
1	0x04
1	region
1	dr
1	tx_power
1	payload_length

6.2.49.6 TST_TX_CW

Transmit a continuous wave.

Table 23 Command payload format for TST_TX_CW command

Size (bytes)	Field
1	0x06
4	frequency
1	tx_power

6.2.49.7 TST_RX_CONT

Continuously receive packets.

Table 24 Command payload format for TST_RX_CONT command

Size (bytes)	Field
1	0x07
4	frequency
1	sf
1	bw
1	cr

6.2.49.8 TST_RX_CONT

Measure RSSI.

Table 25 Command payload format for TST_RX_CONT command

Size (bytes)	Field
1	0x08
4	frequency
2	time_ms
1	dr

The supplied datarate determines the bandwidth.

Table 26 Response payload format for TST_RX_CONT command

Size (bytes)	Field
1	RSSI + 64

6.2.49.9 TST_RADIO_RST

Reset the SX1276 radio.

Table 27 Command payload format for TST_RADIO_RST command

Size (bytes)	Field
1	0x09

6.2.49.10 TST_SPI

Send and receive SPI commands directly to the SX1276 radio.

Table 28 Command payload format for TST_SPI command

Size (bytes)	Field
1	0x0A
var.	spi_command
1	resp_len

Table 29 Response payload format for TST_SPI command

Size (bytes)	Field
resp_len	spi_response

6.2.49.11 TST_EXIT

Exit test mode and reset modem.

Table 30 Command payload format for TST_EXIT command

Size (bytes)	Field
1	0x0B

7 Events

Events are notifications which occur asynchronously to the command-response flow. Pending events are indicated via the EVENT line and can be retrieved via the GetEvent command. Multiple events for different event types might be queued and can be retrieved subsequently. However multiple events for the same event type will overwrite the previous event of that type. For example, if multiple downlinks are received, and a new downlink arrives before the DownData event of the previous one has been retrieved, only the new DownData event can be retrieved. Therefore it is good practice to retrieve pending events as soon as the EVENT line is set. Eventually missed events are indicated in the count field of the response of the GetEvent command.

7.1 Event Overview

Table 31 List of event codes

Name	Code	Description	Data
Reset	0x00	Modem has been reset	reset count[2]
Alarm	0x01	Alarm timer expired	
Joined	0x02	Network successfully joined	
TxDone	0x03	Frame transmitted	status[1]
DownData	0x04	Downlink data received	port[1], downdata[n]
UploadDone	0x05	Fileupload completed	status[1]
SetConf	0x06	Config has been changed by DM	info tag[1]
Mute	0x07	Modem has been muted or unmuted by DM	mute[1]
StreamDone	0x08	Data stream fragments sent	

7.2 Event Details

7.2.1 Reset

This event indicates that the modem has been reset.
It returns the current reset counter which is incremented at every reset. A reset can be caused by UART command, DM request, power failure, or an internal error condition. The application MCU might use this indication to eventually restore some state of the modem (e.g. re-join).

7.2.2 Alarm

This event indicates that the programmed alarm timer has expired.

7.2.3 Joined

This event indicates that the modem has successfully joined the network.

7.2.4 TxDone

This event indicates that the previously initiated RequestTx operation has completed.
It returns a status byte which indicates whether the frame was sent (value 0x01), or if the frame could not be sent because its length exceeded the maximum payload size for the current data rate (value 0x00).

7.2.5 DownData

This event indicates that a downlink with application data has been received.
It returns one byte representing the port of the downlink frame followed by the payload of that frame.

7.2.6 UploadDone

This event indicates the end of a previously initiated file upload.

It returns a status byte which indicates whether the file has been successfully received by the server (value 0x01) or if the upload stream has been aborted because no confirmation has been received from the server (value 0x00).

7.2.7 SetConf

This event indicates that a configuration setting has been changed by the DM service.
It returns one byte holding the tag code of the information field changed. Tags codes are described in section Uplink Message Format.

7.2.8 Mute

This event indicates that the modem has been muted or unmuted by the DM service.
It returns one byte mute status (muted = 0x01, unmuted = 0x00).

7.2.9 StreamDone

This event indicates that the modem has sent all fragments of the previously submitted data record and is now ready to accept new data records.

8 Command Sequence Examples

8.1 Join

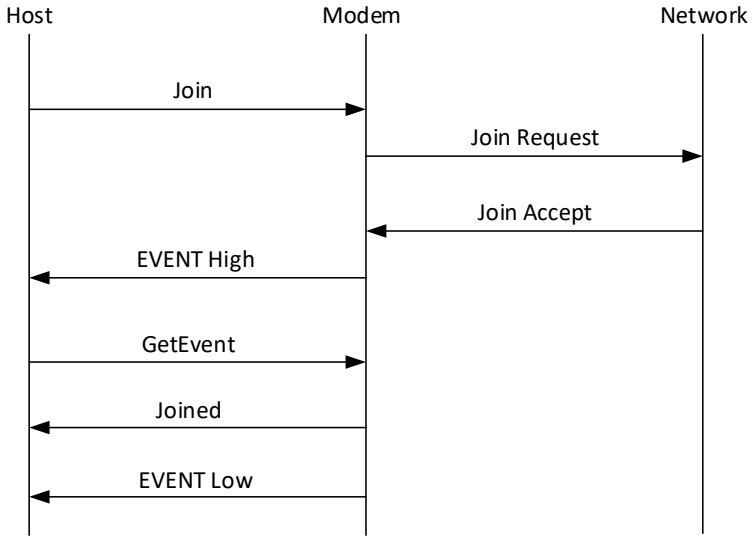

Figure 3 Command sequence for joining the network

8.2 Class A uplink

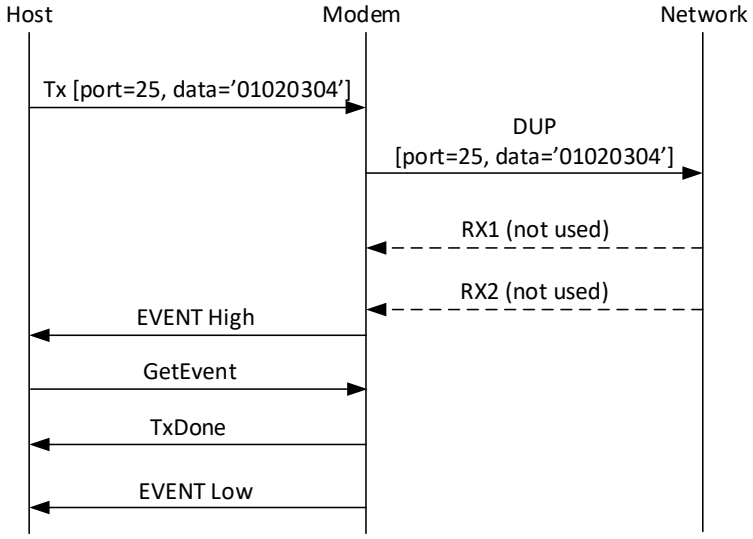

Figure 4 Command sequence for uplink without downlink

8.3 Class A Uplink with Downlink

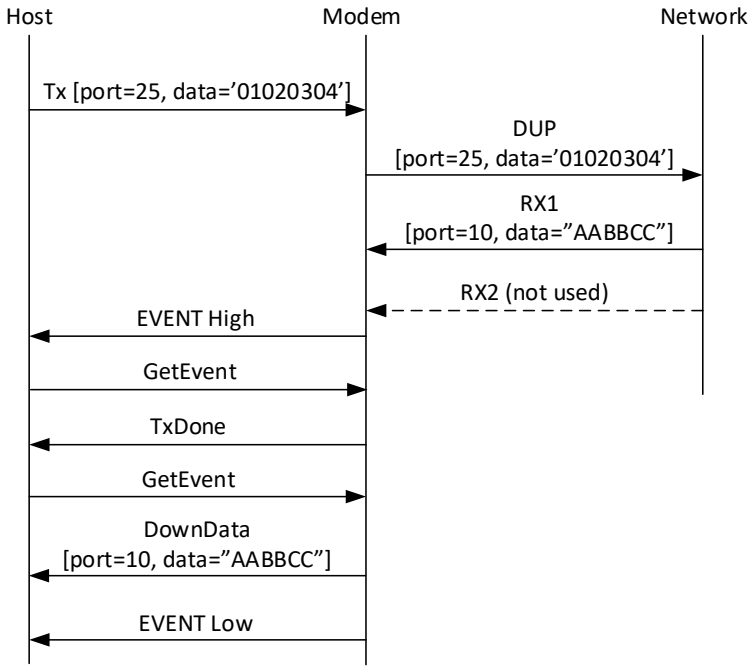

Figure 5 Command sequence for uplink with downlink

9 Modem Service

The Semtech Device Management Cloud Service has an HTTP-based API and is fed with the uplink service messages sent by the modem. In return to each call, it can provide downlink messages to be delivered back to the modem. It also can return defragmented application data records and defragmented application file data.
All uplink messages received on the device management port (default 200) should be forwarded to the cloud service.

9.1 Uplink Message Format

Uplink messages contain one or more concatenated device information fields. All fields begin with one-byte field tag code and are followed by the field information. The content length of most fields is implicit and is defined in the table below. Fields specified as variable length usually have larger content and are sent in separate frames. These fields are not mixed with other fields and the field content length is the full payload length. When due to duty cycle limitations, not all requested fields fit into one message, the fields are split over multiple messages.
All multi-byte integers contained in message payloads are transmitted least-significant-byte-first (LSBF).

Table 32 List of modem service codes

Name	Code	Size (bytes)	Description
status	0x00	1	modem status
charge	0x01	2	charge counter [mAh]
voltage	0x02	1	supply voltage [1/50 V]
temp	0x03	1	junction temperature [deg Celsius]
signal	0x04	2	RSSI [dBm]-64, SNR [0.25 dB]
uptime	0x05	2	duration since last reset [h]
rxtime	0x06	2	duration since last downlink [h]
firmware	0x07	4+2	firmware CRC and fuota progress
adrmode	0x08	1	ADR profile (0-3)
joineui	0x09	8	JoinEUI
interval	0x0A	1	reporting interval [values 0-63, units s/m/h/d]
region	0x0B	1	regulatory region
opmode	0x0C	4	operating mode
crashlog	0x0D	variable	crash log data
upload	0x0E	variable	application file stream fragments
rstcount	0x0F	2	modem reset count
deveui	0x10	8	DevEUI
owncnt	0x11	2	owner number
session	0x12	2	session id / join nonce
chipeui	0x13	8	ChipEUI
stream	0x14	variable	data stream fragments
streampar	0x15	5	data stream parameters
appstatus	0x16	8	application-specific status
alcsync	0x17	variable	application layer clock sync data

9.1.1 Periodic Status Reporting

The modem periodically sends a status message, which by default contains the status, charge, voltage, temp, signal, uptime, and rxtime fields. The default set of fields can be changed by the SetDmInfo modem command or by the SetDmInfo DM request. The first status message after joining additionally contains the rstcount and session fields. Other fields might be explicitly requested using the GetInfo downlink message. The reporting interval can be queried and set using the GetInterval and SetInterval modem commands. Additionally, it can be set by the DM via the SetConf downlink message specifying a value for the interval field.

9.1.2 Format of Reporting Interval

The periodic status reporting interval field is encoded in one byte where the two top-most bits specify the unit (sec/min/hour/day), and the lower six bits the value 0-63. A value of zero disables the periodic status reporting.

Table 33 Format of the reporting interval

Bits	Variable	Description
7-6	Unit	00-sec, 01-day, 10- hour, 11- min
5-0	Value	0-63

9.1.3 Format of Upload Application File Data Fragments

upload fields are of variable length and are sent in a separate message containing only this field.
Each upload field begins with a 16-bit header (LSBF) consisting of 2-bit session id, 4-bit session counter, and 10-bit chunk count-1.
This header is followed by one or more 8-byte chunks of spread file entropy data.

9.1.4 Format of Defragmented Upload Application File Data

When the cloud service has received enough fragments to reconstruct the complete file, it will verify the integrity and will return the file to the application together with a downlink message to indicate stop the streaming of fragments.
This downlink message needs to be delivered back to the device. The file consists of a 12-byte header followed by the file data, which is optionally encrypted.

Table 34 Format of Defragmented upload Application File Data

Header	Size (bytes)	Description
port	1	application port
flags	1	file data is encrypted if non-zero
size	2	size of file data
hash1	4	truncated SHA256 hash over received file data
hash2	4	outer or inner hash over received file data (*)
data	*size*	file data

If no encryption is used, hash2 is the next 32-bit of the SHA256 hash over the received file data. When the file data is encrypted (flags non-zero), hash2 is the truncated SHA256 hash of the plain file data. The decryption of the file data can be performed using the AppSKey for AES in counter mode with the following initial block:

Table 35 AES-CTR initial block

AES-CTR initial block						
Byte Length	1	4	1	4	4	2
Field:	0x49	0x00	0x40	file size (lsbf)	hash2	0x00

After decryption, hash2 can be used to verify the integrity of the plain file data.

9.1.5 Format of stream Application Data Record Fragments

stream fields are of variable length and are sent in a separate message containing only this field. The field payload contains one or more data record fragments as described in **Error! Reference source not found.**.

9.1.6 Format of Defragmented stream Application Data Records

When the cloud service has received enough frames to reconstruct one or more application data records it will return these records retaining the order of transmission. I.e. when a missing record is reconstructed from subsequent redundancy frames, the original order of the returned records is retained.

The rate at which data records can be decoded and returned depends on the selected window size and the packet error rate. Since the returned data records are optionally encrypted, the application might need to decrypt them. The decryption of the record data can be performed using the AppSKey for AES in counter mode with the following initial block.
The AES-CTR initial block is defined as the same as in Table 35.

9.2 Downlink Message Format

The cloud service might return one or more request messages which have to be delivered over the network back to the modem on the device management port (default 200). All downlink messages have the following format:

Table 36 Downlink message format for device management port

Field	Size (bytes)	Description
upcount	1	uplink count
updelay	1	uplink delay [s]
request	1	request code
payload	variable	request data

Next to the request code and data, each message contains an upcount field which indicates the number of uplinks to generate. These uplinks can be used to create additional downlink opportunities and should be generated at the rate specified by the updelay field. The reception of a new request message resets the uplink generation.

9.2.1 Overview of Downlink Messages

The following downlink requests are defined:

Table 37 Downlink request codes for device management port.

Request	Code	Parameters	Description
Reset	0x00	mode[1]+rstcnt[2]	reset modem or application MCU
Fuota	0x01	n	fuota firmware update chunk
FileDone	0x02	sessionidctr[1]	file upload complete
GetInfo	0x03	taglist[n]	get info fields
SetConf	0x04	tag[1]+value[n]	set config field
Rejoin	0x05	sesscnt[2]	rejoin network
Mute	0x06	mute[1]	permanently disable/enable modem
SetDmInfo	0x07	taglist[n]	set list of default info fields
StreamRate	0x08	port[1]+cr[1]	set coding rate for data stream
ALCSync	0x09	n	application layer clock sync data

9.2.2 Downlink Request Details

9.2.2.1 Reset

This request performs a reset of the modem or application MCU (if the expected reset counter matches).
The first parameter indicates which units need to be reset (0x01 = modem, 0x02 = app MCU, 0x03 = both). The second parameter is the expected reset count of the modem. At startup, after reset, the modem generates a Reset event.

9.2.2.2 Fuota

This request delivers a FUOTA firmware chunk to the modem.
In return, the modem will send an uplink with the firmware field indicating the progress of the update. When the modem has received enough chunks to reconstruct the full update, it will automatically reset and install the update.

9.2.2.3 UploadDone

This request signals the successful reception of a file upload in progress.

It has the session id and session counter of the completed file upload as a parameter to identify the completed upload session. The parameter is one byte and is encoded as a 2-bit session id (high-nibble) and 4-bit session counter (low-nibble). The modem should stop streaming file fragments on the reception of this request. An UploadDone event will be generated for the application MCU.

9.2.2.4 GetInfo

This request specifies a list of information tags to be reported by the modem as soon as possible.

9.2.2.5 SetConf

This request allows setting a specific configuration field.
The payload begins with the tag byte of the field to be set and is followed by the value. The length of the value is implicitly defined by the information field. A SetConf event will be generated for the application MCU.

9.2.2.6 Rejoin

This request instructs the modem to rejoin the network.
The parameter is the expected session counter. The modem only rejoins the network if the counter matches, otherwise it will report the current value of the session counter.

9.2.2.7 Mute

When the parameter is non-zero this request will permanently mute the modem.
It will reenable the modem when the parameter is zero. Muting will prevent the modem from transmitting any application messages. Therefore, modem commands which would trigger transmissions will fail with Fail and Mute status. A Mute event will be generated for the application MCU indicating the current mute state in the status byte. When the parameter is non-zero, its value specifies the number of days when the modem will send a status message anyway (0xFF never).

9.2.2.8 SetDmInfo

This request specifies the list of information tags to be included in the periodic status messages.
By default, the status, charge, voltage, temp, signal, uptime and rxtime fields are reported.

9.2.2.9 StreamRate

This request sets the new coding rate for the data stream on the specified port.

9.2.2.10 ALCSync

This request provides time correction feedback for the application layer clock sync protocol.

10 Demonstration Kits

10.1 STM32 Discovery Kit

For development purposes, the modem is available on the STM32 Discovery Kit platform.

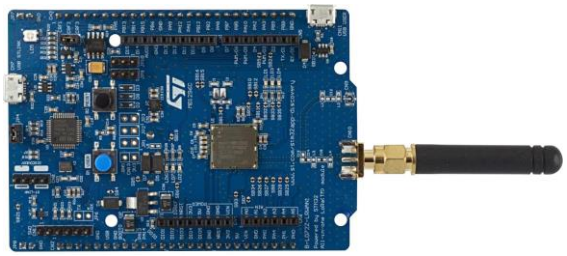

Figure 6 STM32 Discovery Kit platform

10.1.1 Serial interface

On the B-L072Z-LRWAN1 discovery board, the Serial Interface of the modem is accessible through the following pins.

Table 38 Interface to B-L072Z-LRWAN1 discovery board

Function	STM32 GPIO	CN3 header
UART RX	PA10	Pin 21
UART TX	PA9	Pin 13
COMMAND	PB8	Pin 25
BUSY	PA8	Pin 18
EVENT	PB9	Pin 24
MCURESET	PB2	Pin 17

10.1.2 FTDI cable connection

A development host PC can be conveniently connected to the modem via an FTDI (USB to serial) adapter cable. The table below shows how the TTL-232R-3V3-2mm FTDI cable needs to be connected to the CN3 header pins of the discovery board.

Table 39 FTDI cable connection

Modem Function	CN3 header	FTDI cable
COMMAND	Pin 25	RTS (green)
UART TX	Pin 13	RXD (yellow)
UART RX	Pin 21	TXD (orange)
–	–	VCC (red)
BUSY	Pin 18	CTS (brown)
GND	Pin 12	GND (black)
EVENT	Pin 24	–
MCURESET	Pin 17	–

10.1.3 USB Connection

Additionally, the discovery kit can provide a virtual serial port over USB to the development host PC to print diagnostic output. To view the diagnostic output a terminal application needs to connect to the serial port with 115200bps / 8N1 parameters.

10.2 Nucleo Shield Kit (preliminary information - under development)

For easy hardware integration, a modem shield is designed.

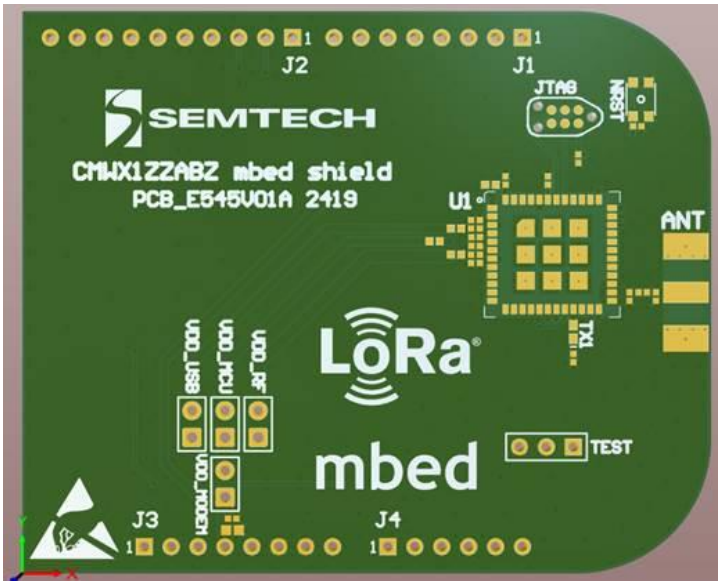

Figure 7 Modem shield for Nucleo board

Table 40 Pin connections for modem shield

Modem Function	CN8 header	CN5 header
UART TX	Pin 2	
UART RX	Pin 1	
BUSY		Pin 5
COMMAND		Pin 4
EVENT		Pin 3
RESET		Pin 2

Developers or users can use the board to develop applications on nucleo boards.

11 Reliable Octet Stream Encoding (ROSE)

11.1 ROSE Concepts

This protocol transports a stream of octets from a LoRaWAN device to a service endpoint in the cloud. The transport protocol ensures high reliability of message delivery by sending additional redundancy information.
Conceptually, the octet stream is like a file where the device appends data and a server-side application reads data once available. All octets in the stream have a unique offset like a byte in a file. The first octet has offset 0 and all data subsequently submitted by the device gets incremental positions.
The octet stream protocol has a number of parameters, which can be controlled from the server side. This enables the server to do data analysis and optimize the trade-off between message overhead, latency, and reliability according to application-specific needs.

11.1.1 Server side view of a Reliable Octet Stream

The server presents the data stream as a sequence of elements. Each of the elements has a unique position index, called stream offset. This offset starts at zero with the first octet being transmitted in each join session and it is strictly increasing during the join session. An element either carries a value 0..255, representing a transmitted octet, or a state indicating one of the following two conditions:

- The octet might be recovered from impending redundancy data.
- The octet is definitely lost due to high packet loss or due to buffer overrun on the device side.

The following table shows an example of data presented to an application on the server side. The application has already consumed data from offsets 0 to 0x344F and the table displays unconsumed data from position *0x3450+0x00 to 0x3450+0xBC*.

Table 41 Example of ROSE server-side data

```
streamOffset=0x3450

0x00: xxxxGA,123529,2307.038,N,01131.0
0x20: 00,E,1,08,0.9,545.4,M,46.9,M,,*4
0x40: 7..$GPGSA,A,3,04,05,,09,12,,,24
0x60: ,,,,,2.5,1.3,2.1*39..$?PGSV,2,1
0x80: ,08,01,40,083,4??2,17,3?8,41,12
0xA0: ,07,344,39,14,2?,228,45*75..
```

The symbol ? marks an octet that is not yet known and has a non-zero chance of still being recovered. The symbol x marks an octet that is definitely lost. The server buffers a certain amount of data and an application consumes data on a regular basis. In the example above the application might consume offsets *0x3450+0x00 .. 0x3450+0x75*, two NMEA sentences, and might wait for the last sentence to be completed eventually.
Finding message boundaries in the octet stream is application specific. There are two approaches: delimiters or fixed record sizes. The example above uses delimiters. The *$* or the *CR/NL* can be used to find data records in the stream.
Another method is using fixed record lengths. Instead of transmitting NMEA sentences, an application could send a binary record with latitude/longitude/altitude. Such an octet stream could look like this:

Table 42 Example of ROSE server-side data

```
streamOffset=0x3450

#     <---lat---> <---lon---> <alt>
0x00: xx xx 52 FC 05 42 87 E0 01 A4
```

```
0x0A: 1C 26 52 D0 05 42 87 E8 01 A5
0x14: 1C 26 52 38 ?? 42 87 ?? 01 A7
0x1E: 1C 26 52
```

An application finds the start of a record at every stream offset being a multiple of 10 in this example.
Applications may choose to drop any extracted record that is damaged, e.g. contains any x elements. It may choose to ignore records that contain **?** elements and skip ahead to completed records further down the stream and consider or ignore older records once they are completed. This leads to out of order processing of records by but might be desirable to minimize latency. If out of order processing is not possible then it must wait until **?** elements are resolved into octets or are marked as lost (x) in order to process records strictly in order.
In addition, another layer of recovery may be applied to incomplete data records. Some data may still be recovered by considering application-specific knowledge. For instance, certain positions in NMEA sentences may not contain valuable information and some positions might be reconstructed from the context of previous records. If the most significant digit of latitude is lost it could be inferred from previous positions assuming the device may not have moved this much. If the least significant digit of latitude is lost, the remaining data may still be used by decreasing the accuracy of the position.
Having partial record data enables an application to apply more sophisticated recovery attempts on a semantic level.

11.1.2 Device Side View of a Reliable Octet Stream

The octet stream abstraction presents itself on the device side as a FIFO. The streaming application on the device can add any amount of data to the FIFO at any time. The FIFO should have a size big enough to hold the redundancy window (WL), some small overhead required by the protocol, and a buffer to accommodate data arriving under peak load until it is drained.
The octet stream protocol engine takes the buffered data and sends it first as systematic data (as plain text) and then again as part of redundancy data. Redundancy data is a random mix of the previous N systematic octets. The parameter N is called the window length and defines how far the redundancy algorithm looks back. The window length limits the maximum amount of consecutive data loss the algorithm can handle. On the flip side, longer window lengths imply a bigger latency when data needs to be recovered.
The application may put data into the FIFO buffer until it is full. Draining the FIFO may take some time because a number of LoRa frames need to be sent. If the FIFO is full the application has two choices. It can drop the new data and, if possible, suspend data generation via flow control towards the data source until there is space again in the FIFO. Or it could add the data to FIFO anyway which leads to an overrun. This forcefully displaces older data from the FIFO which has either not been sent at all or has not yet reached the targeted redundancy rate. The server will perceive this as packet loss and will either recover some data or mark octets as lost. In any case, the stream offset is maintained properly even under buffer overrun conditions.
Of course, a FIFO overrun should be a rare event, probably happing only under peak load conditions. Otherwise, the system is conceptually overloaded and will only work in a degrading way.

11.1.3 Data Units Other Than Octets

So far, the protocol assumed the basic data unit is an octet. **ROSE** allows to customize this and use 16 bit, 32 bit or even 64-bit data units instead of just 8 bit. Using bigger data units has the advantage of reduced compute overhead and state sizes on the server side. The disadvantage is that data can only be sent in multiples of the basic data unit size. Applications must either insert padding or wait until enough information is available. In addition, LoRaWAN frames cannot be utilized for bigger data unit sizes and small LoRaWAN frame lengths.
Both device and server must be in agreement over the size of the basic data unit. It is not a negotiable protocol parameter that can change during a session. The device indicates the data unit size in use in a **SINFO** message in case the server is in doubt over this value.

11.2 Protocol Messages

The protocol can be mapped to an arbitrary LoRaWAN port. Both server and device must agree on this specific LoRaWAN port to run the **ROSE** protocol. Separate instances of the protocol can be run on different LoRaWAN ports in parallel provided the device has enough resources to manage the associated state.
There are two uplink messages **SDATA** and **SINFO** and one downlink message **SSCMD**. Under normal operating conditions there are only **SDATA** messages sent by the device. The messages **SSCMD** and **SINFO** are only required if the server wants to learn or change some protocol parameters on the device.

- **Note:** Multi-byte fields treated as integer values are stored in little-endian byte order in the same way as similar fields in the LoRaWAN specification.

11.2.1 Streaming data message (SDATA)

A device uses this message to forward octet information to the server. An SDATA message carries two types of data: systematic octets (data in clear) and redundancy octets (the combination of previously sent systematic octets). The message can carry only systematic data (M>0, N=0), only redundancy data (M=0, N>0), or both (M>0 and N>0).

Table 43 Byte layout of an SDATA message

Byte Length	**1**	**2**	**N**	**M**	**1**	**2**
Field:	SHDR	SOFFL	SYSDAT	REDDAT	WL	SOFFL

The bit layout of the SDATA.SHDR field:

Table 44 Bit layout of the SDATA.SHDR field

Bits	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Field:	PCTX	SYSC						

Description of the fields and bits:

- **SHDR.PCTX**
  This flag indicates the absence/presence of the fields **WL** and **SOFFH**. The device will enable this bit from time to time so that the server can learn these protocol parameters. Once learned, the server can track these values automatically.
- **SHDR.SYSC**
  The header byte specifies the length of the systematic bytes. The value of **SYSC** may be in be the inclusive range 0 .. 2^7-2. The value 2^7-1 is not allowed to disambiguate **SDATA** messages from **SINFO** messages.
- **SOFFL**
  The least significant 16 bit of the stream offset **SOFF**.
- **SYSDAT**
  The systematic data octets. **SYSC** == M. It is legal for an **SDATA** message to contain zero systematic data octets. The first octet in this field has the offset expressed in the corresponding fields **SOFFL** and **SOFFH**.
- **REDDAT**
  Redundancy data octets. The field length M *is* calculated as $FRMPayload - N$ - 3 - 3 * **PCTX**. M may be zero. The redundancy octets are a combination of the systematic octets immediately preceding the stream offset as expressed in the corresponding fields **SOFFL** and **SOFFH**.
- **WL**
  The current window length of the redundancy algorithm. The redundancy algorithm considers the **WL** octets preceding the current stream position. `Note:` This field is only present if **PCTX** is 1.
- **SOFFH**

The most significant 16 bits of the stream offset **SOFF**.

`Note:` This field is only present if **PCTX** is 1.

If the server receives **SDATA** messages and has no clue about the values of **WL** and **SOFFH** then it should buffer these messages until an **SDATA** with **SHDR.PCTX** set is received. At that point, the server can replay the backlog of messages considering the just discovered values.

11.2.2 Streaming synchronization message (SINFO)

A device reports all protocol parameters by sending **SINFO** messages. This kind of message is either sent because the server asked for the current parameters (**SCMD.FLAGS.SINFO**) or because the server changed the window length (**SCMD.FLAGS.UPDWL**).

Byte layout of **SINFO** messages:

Table 45 Byte layout of SINFO message

Byte Length	**1**	**1**	**1**	**1**	**2**	**2**	**1**
Field:	**SHDR**	**FLAGS**	**WL**	**RR**	**SOFFL**	**SOFFH**	**PCTX INTV**

The bit layout of field **SINFO.FLAGS**:

Table 46 Bit layout of field SINFO.FLAGS

Bits	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Field:	**RFU**					**USZ**		**RQAWL**

Description of the fields and bits:

- **SHDR**
  This field must have value $0xFF$. This value disambiguates **SINFO** from **SDATA** messages.
- **FLAGS.RQAWL**
  If **RQAWL** bit is clear then this is just an informal message to report the current protocol parameters. If **RQAWL** bit is set then the device requests an acknowledgment from the server to confirm the changed window length. The server must respond with an **SCMD** message with bit **ACKWL** set. Once the device receives this acknowledgment it will resume sending **SDATA** messages. Until then **SINFO** messages are being sent to obtain an acknowledgment.
- **FLAGS.USZ**
  This field specifies the unit size of data items. The size in bytes of a data item is expressed as 2^{USZ}. The stream operates on octets if USZ=0.
- **WL**
  The current window length.
- **RR**
  The current redundancy rate. For every systematic octet, the device sends approximately **RR**/100 redundancy octets.
- **SOFFL / SOFFH**
  The current stream offset. The next systematic octet being sent will have this offset. The next redundancy octets being sent will be constructed from octets in the inclusive range **SOFF** - 1 to **SOFF** - **WL**.
- **PCTXINTV**
  The device enables the **SHDR.PCTX** flag in **SDATA** messages after this number of messages without.

11.2.3 Streaming command message (SCMD)

This message is sent by the server to the device to query or change protocol parameters.

The byte layout of an **SCMD** message:

Table 47 The byte layout of an SCMD message

Byte Length	1	1	1	1
Field:	FLAGS	WL	RR	PCTXINTV

The bit layout of field **SCMD.FLAGS**:

Table 48 Bit layout of field SCMD.FLAGS

Bits	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Field:	RFU		UPDPCI	UPDRR	UPDWL	ACKWL	SINFO	SCMD

Description of the fields and bits:

- **FLAGS.SCMD**
  This bit must be set to 1 to distinguish **SCMD** data from other downlink data forwarded to the application feeding the stream.
- **FLAGS.SINFO**
  If set the device shall be sent a **SINFO** message.
- **FLAGS.ACKWL**
  If set the server acknowledges the reception of a **SINFO** message with bit **ACKWL** set. The **WL** field contains the acknowledged window length. The device ignores this message if the **WL** does not match the current window length. If this bit is set **FLAGS.UPDWL** must not be set.
- **FLAGS.UPDWL**
  If set the server asks the device to change the window length, the device will stop sending **SDATA** messages and instead send **SINFO** message with **SINFO.FLAGS.ACKWL** set to 1 and the new window length. The device will continue sending **SDATA** once the server has acknowledged this **SINFO** message. If this bit is set **FLAGS.ACKWL** must not be set.
- **FLAGS.UPDRR**
  If set the device changes the redundancy rate to the value contained in the field **RR**.
- **FLAGS.PCI**
  If set the device changes the protocol context interval to the value contained in the field **PCTXINTV**.
- **WL**
  The new window length if **FLAGS.UPDWL** is set or the acknowledgment of the window length change if **FLAGS.ACKWL** is set. This field is ignored if neither **FLAGS.UPDWL** nor **FLAGS.ACKWL** are set.
- **RR**
  The new value of the redundancy rate. Valid only if **FLAGS.RR** is set.
- **PCTXINTV**
  The new protocol context interval. Valid only if **FLAGS.PCI** is set.

11.2.4 Streaming Offset (SOFF)

The streaming offset tracked by the protocol has a size of 32 bits. The least significant 16 bits **SOFFL** are included in each **SDATA** message. The most significant bits **SOFFH** are included in **SDATA** messages every now and then if the **SDATA.SHDR.PCTX** bit is set.

Table 49 Bit layout of SOFF

Bits	**31-16**	**15-0**
Field:	SOFFH	SOFFL

The stream offset starts at zero with the first octet submitted by the device to the protocol engine. All subsequently submitted data gets contiguous increasing offset numbers assigned. After the device rejoins the streaming offset restarts at zero.

If **SDATA.SOFFL** rolls over the server can either wait for the next **SDATA** with protocol context information or safely assume a single rollover if the difference of FCntUp values makes a double rollover impossible. Let D be the difference of FCntUp values between the current and the last **SDATA** frames. If $(D-1)*2^7-1$ is smaller than 2^{16} then there could have been only a single rollover and the server can safely track this change without waiting for a message with protocol context.

11.2.5 Encoding of Window Lengths

The window length is a parameter of the redundancy algorithm. Both server and device must be in agreement over this parameter. If not in sync, the server might reassemble garbage from redundancy data without being able to detect this situation.

An 8-bit value is used to encode various reasonable length values.

Table 50 Bit layout of window length

Bits	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Field:	WLSC		WLPF					

The four values of the size class **WLSC** select a specific formula to calculate the **WL** in octets:

Table 51 WLPC and WL calculation

WLPC	**WL formula**
0	16 + 4 * WLSF
1	272 + 8 * WLSF
2	784 + 16 * WLSF
3	RFU

11.2.6 Construction of Redundancy Data

Redundancy data is constructed from the last N systematic octets that have been transmitted. The protocol parameter N is called window length (**WL**). Both server and device use a pseudo-random number generator (PRNG) to select a set of octets by chance from the redundancy window. The PRNG is seeded by the information available to both, the server and the device. Thus, no extra information needs to be transmitted.

If we assume an **SDATA** message with K octets of redundancy information which is to be sent in a Lo- RaWAN frame with frame counter $FCntUp$. Let i be the position of the redundancy octet in the field **SDATA.REDDAT** and $W[x]$ the xth octet in the redundancy window. If **SOFF** is the current stream offset then $W[0]$ is the octet with stream offset **SOFF** - **WL**. Each of the redundancy octets $reddat[i]$, with $0 \leq i < K$, is constructed in the following way:

Table 52 Example code for constructing the redundancy octets

```
S = emptySet
while len(S) < WL/2:
   S += PRNG(FCntUp, i, WL)

red(i) = XOR( W(k) for all k in S )
```

Table 53 Example code for PRNG algorithm

```
def PRNG(FCntUp, i, WL):
   wlfixed = WL + int((WL-1) & WL
   == 0) x = 1 + 1001 *
   (FCntUp ^ (i<<8))
   r = 1 << 16
   while r
      >= wl:
      x=
      pbrs23
      (x)
      r = x   wlfixed
   return r
```

Important Notice

Information relating to this product and the application or design described herein is believed to be reliable, however such information is provided as a guide only and Semtech assumes no liability for any errors in this document, or for the application or design described herein. Semtech reserves the right to make changes to the product or this document at any time without notice. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. Semtech warrants performance of its products to the specifications applicable at the time of sale, and all sales are made in accordance with Semtech's standard terms and conditions of sale.

SEMTECH PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS, OR IN NUCLEAR APPLICATIONS IN WHICH THE FAILURE COULD BE REASONABLY EXPECTED TO RESULT IN PERSONAL INJURY, LOSS OF LIFE OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. INCLUSION OF SEMTECH PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE UNDERTAKEN SOLELY AT THE CUSTOMER'S OWN RISK. Should a customer purchase or use Semtech products for any such unauthorized application, the customer shall indemnify and hold Semtech and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs damages and attorney fees which could arise.

Contact Information

Semtech Corporation
Wireless & Sensing Products
200 Flynn Road, Camarillo, CA 93012
E-mail: sales@semtech.com
Phone: (805) 498-2111, Fax: (805) 498-3804

www.semtech.com