Interfacing with LoRaWAN

RM186 LoRa + BLE Module

Application Note *v1.3*

INTRODUCTION

The Laird RM1xx is a series of wireless communications modules that combines a Nordic nRF51822_QFAC (256/16) BLE device and a Semtech SX1272 860 to 1020 MHz low power, long range transceiver. The RM186 and RM186_PE are designed to function in the EU863-870MHz ISM band as opposed to the RM191 and RM191_PE, which is a similar device designed for the US 902-928 MHz ISM band.

Both versions can collect data from external sources by interfacing directly with a sensor over a UART, SPI or, I^2C serial communication channel; or over a wireless BLE connection if a sensor is physically connected to a BLE peripheral device. Data can also be gathered internally using the analog or digital IO pins. Once the data has been collected and arranged in packets, it can then be transmitted to a remote LoRaWAN Gateway up to 16 kilometers away. From there, the data can be transmitted to a server or database over a TCP/IP link.

As well as supporting the two distinct frequency ranges, the LoRaWAN specification also supports two protocols. Although there are slight differences relating to power levels and data rates, the main differences between these protocols are how a module joins a network; specifically which frequency it must use and how the network then configures the module once it has joined.

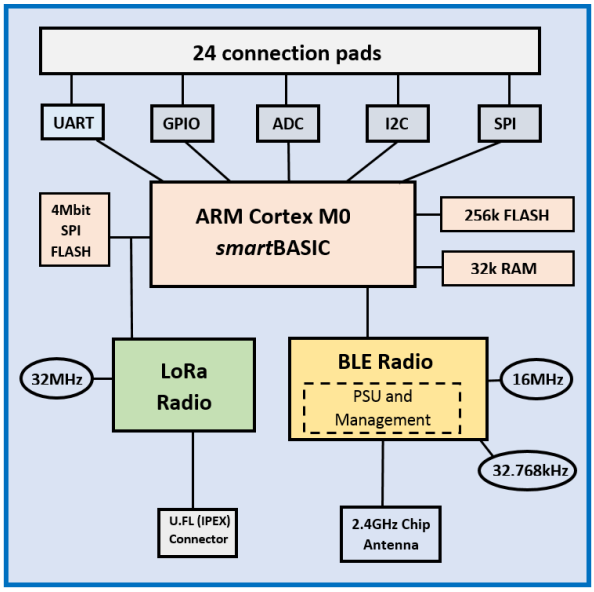
The first protocol, used in the EU and AS (Asia) regions, requires that the modules support a specified number of mandatory channels on which a JoinRequest must be transmitted. Then, as part of a JoinAccept message, the network configures up to five additional channels on which that the module can then transmit data packets.

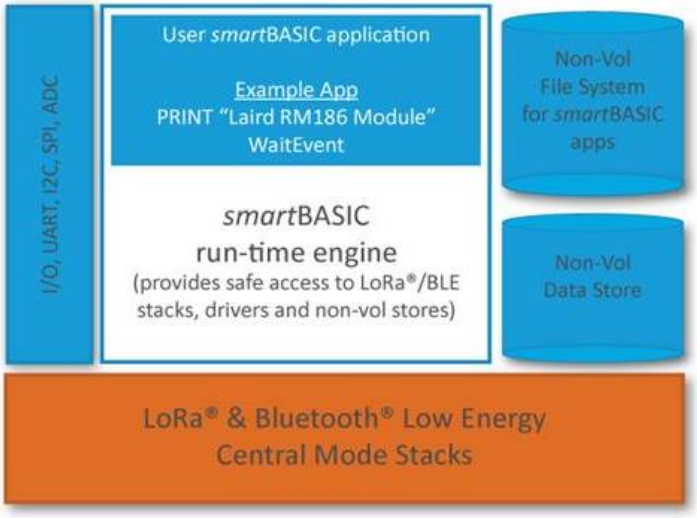
The second protocol, used in the US and AU regions, initially enables 64 125 kHz and eight 500 kHz channels and transmits JoinRequests on random alternate 500 kHz and 125 kHz channels until it transmits on a frequency supported by a receiving gateway. Once the module joins the network, the network can then update the enabled channels by means of a specific MAC command which modifies an internal ChannelsMask parameter. This tracks the state of the channels.

This document describes how the RM186 and RM186_PE, which support the EU region protocol, interface with the LoRaWAN specification. It also explains what happens inside the module to enable support for this protocol.

All *smart*BASIC APIs and LoRa Events referenced below are described more fully in the *RM1xx Series smartBASIC LoRa Extensions* user guide. This guide is available from the Documentation tab of the RM1xx Series product page.

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610





GATEWAYS AND SERVERS

LoRaWAN is a wireless standard developed by the LoRa Alliance. It enables low-power wide-area networking using low data rates and minimal power usage. The RM1xx modules are designed to communicate with any LoRaWAN gateway that is within RF range, can receive a signal within the correct frequency range, and meets the requirements defined in the appropriate section of the LoRaWAN specification.

The gateway does nothing with the data, except to time stamp it and then send it to a network server via a TCP/IP link. The network server must be running LoRaWAN-compatible software to handle the decryption of messages and authentication of devices. There are several vendors that provide such services.

A module must be set up as a member of this network. This involves a network supporting the application that the module provides, so the network server knows which Application server to send the data to, and the Application server having a list of all the modules that support that application.

It is possible that a packet could be received by a LoRa gateway that is not connected to the desired network. However, in this instance the data should be discarded by the network server as it cannot be decrypted correctly. The only network server that would be able to decrypt the data successfully is the one to which the RM186 is a member. This is explained in greater detail in the Connecting to the LoRaWAN Network section.

TRANSMIT/RECEIVE SEQUENCE

Before looking at the various commands and events it is very important to understand that once you send a packet into the LoRa stack you are starting a predefined sequence that must be allowed to complete before you can send another packet.

In loRa an uplink packet refers to a packet transmitted by the LoRa module and a downlink packet refers to a packet transmitted by the network server/Lora Gateway.

In Class A devices, once an uplink packet is transmitted, the module waits a predefined time for a downlink packet. This packet may or may not be transmitted by the gateway, however, the module must still open up receive windows to receive it if necessary.

For unconfirmed packets this is the end of the cycle. There are no resends. The module assumes that the transmitted packet has been received at the gateway and onto the server. The server may have data to send back to the module and these should be picked up in whichever Receive window they coincide with and dealt with accordingly by the stack.

However, in the case of confirmed uplink packets the cycle is more complicated as the stack actively waits or an acknowledgement. If that acknowledgment isn't received the stack automatically resends the packet a set number of times. The sequence only finishes after an acknowledgement is received or all the retries are attempted or an error has occurred causing the sequence to abort.

Similarly, for the JoinRequest, if the stack fails to receive a JoinAccept, it resends the JoinRequest a set number of times. Again, the sequence only finishes after a JoinAccept is received or all retries are attempted or an error has occurred.

This can be very offputting in the RM1xx because there is no indication of all that was happening in the background; it could seem that the module had frozen. However, this is not the case so it is very important not to interfere with this sequence by sending another packet to the stack.

The events that mark the end of the sequence are indicated in section 5.3. It is very important to monitor all these sequence complete events and act on them accordingly.

For testing purposes, it is possible to use the LoramacSetDebug command or monitor the EVLORAMACTXDONE or LORAMACNOSYNC events. These should reassure that there is actually something happening in the background. In the case of the US/AU modules this should be quite quick as there are no duty cycle restrictions. In the case of the EU modules, however, it is likely to be minutes between resends. However here you can use the LoramacGetOption(LORAMAC_OPT_NEXT_TX) command to check if there is another packet due for transmission.

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0600

EU Frequencies and Duty Cycles

The RM186 module functions in the EU863-870 MHz ISM band, which is spilt up into several bands by the governing ETSI standard. Each band has its own power and duty cycle requirements.

If a band uses up its duty cycle allowance, no frequency channels within that band can be used to transmit a packet. However, that packet can still be transmitted on a frequency channel in another band, provided that band has available duty-cycle. It is very important to understand that the duty cycle belongs to a band and not to each individual frequency channel within that band.

The sub-bands used in the RM186 are shown in the Table 1.

Table 1: RM1xx channel frequencies

Band	Edge Frequencies (MHz)		Duty Cycle (%)
0	865	868	1
1	868	868.6	1
2	868.7	869.2	0.1
3	869.4	869.65	10
4	869.7	869.7	1

It is mandatory for a LoRaWAN end-device to support the following three Band 1 frequencies: 868.1, 868.3, and 868.5MHz. These frequencies are used during the OTAA join procedure which is described below.

Additional frequencies can be added by the LoRaWAN gateway/server in any of the previously-defined bands.

If the RM186 successfully joins a network using OTAA, the LoRaWAN gateway/server transmits a JoinAccept message which, if the network server meets the LoRaWAN spec, should include a list of up to five additional frequencies on which the module could then transmit.

The LoRaWAN gateway/server could also configure additional frequencies using the LoRaWAN **NewChannelReq** command. This command, with the exception of the three mandatory channels mentioned above, could also modify existing frequency channels. Both of these actions are invisible to the user; they occur purely at the LoRaWAN stack level and no notification is provided regarding any changes.

However, it is possible to view a list of the available frequencies using the following *smart*BASIC command:

LORAMACGetOption(LORAMAC_OPT_CHANNELLIST,var$)

This is a read-only command and is documented in the RM1xx LoRa *smart*BASIC extensions manual.

Note: This command only returns the list of enabled channels. There may be more channels configured, but if they are not enabled by the server through the **ChannelsMask** parameter then they are not listed. During the OTAA process, the **ChannelsMask** is configured such that only the three mandatory Band 1 frequencies are enabled.

After joining, the ChannelsMask value can be modified as part of the LoRaWAN **JoinAccept, NewChannelReq**, and **LinkAdrReq** commands. Again, the user is not notified of any changes, however the current value of the ChannelsMask can be obtained through the following command:

LORAMACGetOption(LORAMAC_OPT_CHANNELMASK,var$)

As with the channel list, this is a read-only command.

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0600

Note: More details regarding all *smart*BASIC commands referenced in this application note can be found in the RM1xx *smart*BASIC Extensions Guide, found in the documentation tab of the RM1xx Series product page.

As well as the default frequency channels, there is also a requirement for a default second receive window frequency at 869.525 MHz. The purpose of this frequency channel is explained in the Data Reception below. It is a receive-only frequency and cannot be used to transmit data.

LoRaWAN Events

A series of events are raised by the LoRaWAN stack to indicate the success or failure of a specific task. These events are then routed through the RM186 firmware and output as *smart*BASIC events. Some of these events are raised by the stack. Some have been created by Laird to make the system slightly easier to use.

The events listed in **Error! Reference source not found.** are referenced later in this document.

Table 2: LoRaWAN events

Event	Description
EVLORAMACJOINING	A JoinRequest has been sent to the gateway/server.
EVLORAMACJOINED	The JoinRequest has been successfully received by the gateway/server and the subsequent JoinAccept has been received by the RM191. The module is now connected to the network.
EVLORAMACJOINFAIL	Indicates that a JoinAccept message contains a MIC error. However, no direct action should be taken on receipt of this event.
EVLORAMACTXDONE	A signal from the module indicating that a packet was transmitted from the SX1272. This can be a JoinRequest, LinkCheck or data packet. This is an important event as all subsequent receive timings are taken from this event. However, no direct action should be taken on receipt of this event.
EVLORAMACTXCOMPLETE	Created when an uplink packet is loaded into the radio. However, this event is not thrown until the successful end of the uplink/downlink sequence.
EVLORAMACRXDATA	A downlink packet was received that contains data from the server to the module.
EVLORAMACRXCOMPLETE	A downlink packet was received by the module. This event can indicate a combination of an acknowledgement, a Mac command, or actual data from the server. Downlink packets are not limited to confirmed uplink packets. Mac commands and/or downlink data packets can be transmitted in response to an unconfirmed uplink packet.
EVLORAMACTXTIMEOUT	Very rarely thrown. Usually indicates that a packet failed to be transmitted due to an internal SPI error.
EVLORAMACRXTIMEOUT	Only valid with JoinRequests, confirmed uplink, or LinkCheck packets. For joinRequests and confirmed uplink packets, it is sent after the module fails to receive a downlink after the configured number of transmission attempts. A LinkCheck is not retransmitted so is sent if the LinkCheck doesn't receive a response.
EVLORAMACRXERROR	An RxError was received from the SX1272.

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0600

Event	Description
EVLORAMACSEQUENCECOMPLETE	This event is an amalgamation of all the events that can indicate the end of an uplink/downlink sequence. Two variables are returned with this event. The first indicates the terminating event with the following values: 1: **EVLORAMACTXCOMPLETE** 2: **EVLORAMACRXCOMPLETE** 3: **EVLORAMACRXTIMEOUT** 4: **EVLORAMACRXCRCERROR** 5: **EVLORAMACTXTIMEOUT** 6: **EVLORAMACRXERROR** 7: **EVLORAMACTXDRPAYLOADSIZEERROR** The second parameter is the time until there is available duty cycle to transmit another uplink packet. This is also the time until the **EVLORAMACNEXTTX** Event is thrown. For modules that do not have duty cycle constraints, this is always 0.
EVLORAMACNEXTTX	This event indicates that there is now duty cycle available to transmit the next uplink packet. For modules that do not have duty cycle constraints, this event is always thrown at the same time as the **EVLORAMACSEQUENCECOMPLETE** event.
EVLORAMACNOSYNC	Notification that a receive window closed without receiving a sync pulse. This event is for information only. No action should be taken on receipt of this event.
EVLORAMACADR	Notification that an ADR command was received from the gateway. Some transmit parameters may have changed. This event also returns 2 parameters - Packet Type: 3: unconfirmed downlink packet 5: confirmed downlink packet Frame Pending : If 1 it indicates that the server has any packets yet to be transmitted.
EVLORAMACLINKCHECKRESPMSG	Notification that the module received the response to a LinkCheck packet.
EVLORAMACMICFAILED	There was a MIC error in a received downlink packet.
EVLORAMACDOWNLINKREPEATED	The received downlink packet is a repeat of the previous downlink packet.
EVLORAMACTXDRPAYLOADSIZEERROR	Notification that a packet is too large to be transmitted with the current datarate setting. This event can only occur during the resending of a packet where the datarate has been reduced by the stack.
EVLORAMACFRAMESLOSS	Indicates that the received downlink sequence number is larger than the maximum expected value. This event should not be acted upon directly, the stack should correct this automatically.

DATA RATE

The LoRaWAN specification states that the following data rates must be supported by the RM186.

Note: These figures are specific to CE operation with the RM186. The FCC rules and the RM191 vary.

Table 3: LoRaWAN data rates

Data Rate	Configuration	Physical Bit Rate (bit/s)
0	SF12 @ 125kHz	250
1	SF11 @ 125kHz	440
2	SF10 @ 125kHz	980
3	SF9 @ 125kHz	1760
4	SF8 @ 125kHz	3125
5	SF7 @ 125kHz	5470
6	SF7 @ 250kHz	11000
7	FSK:50kbps	50000

SF7 to SF12 refer to the spreading factor of the system which is defined as the number of chips of information representing each bit (or symbol) of payload data. The actual chip value is calculated by 2^x. For SF7, there are 2^7 (128) chips per symbol and for SF12 there are 2^{12} (4096) chips per symbol.

This explains why the physical bit rate is directly related to the spreading factor. The higher the spreading factor, the more chips that must be sent for each bit of information and so, consequently, the longer it takes to transmit the data. This is very important when you look at the potential data throughput of a module.

At data rate 0, it takes approximately 1.6 seconds to transmit a packet with a 16-byte payload. This means that in a 1% band, you are unable to send another packet in that band for approximately 160 seconds.

At data rate 5, that same packet takes approximately 66 milliseconds to transmit. Therefore, the module is able to transmit another packet at that frequency or another frequency in that band after about 6.5 seconds.

The default configured data rate for the RM186 is data rate 0, SF12@125kHz. At any time, you can reconfigure this value using the **LORAMACSetOption(LORAMAC_OPT_DATA_RATE,var$)** command and read it back using the **LORAMACGetOption(LORAMAC_OPT_DATA_RATE,var$)** command. These are important commands as the server can modify the data rate at any time using the **LinkAdrReq** command.

As with any changes to the configuration of the frequency channels by the gateway/server, notification of any changes to the data rate are not automatically passed back to the user. However the **EVLORAMACADR** is thrown on receipt of an ADR command from the server; you can then call the appropriate *smart*BASIC **LORAMACGetOption** commands to obtain the latest configuration.

Note: Increasing the data rate reduces the maximum range of the module by making the signal more susceptible to noise. At high spreading factors, the receiver can receive data at much lower signal strengths than it can with low spreading factors.

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0600

CONNECTING TO THE LORAWAN NETWORK

Before the RM186 can transmit any data to a LoRaWAN network it must first be connected to that network. As the RF link between the RM186 and a gateway is over a secure channel, this connection process involves the exchange of certain keys between the module and the network server.

This can either be achieved by calculating the keys afresh every connection or by configuring the RM186 and gateway server with the key information in advance.

In both cases below, the module throws the **EVLORAMACJOINING**, **EVLORAMACJOINED** and **EVLORAMACNEXTTX** during the Join process.

Over-the-Air Authentication

The recommended method of connecting an RM186 to a network is by using the Over-the-Air Authentication (OTAA) method. With this method it must be configured with certain IDs so that the Network Session Key (NwkSKey) and the Application Session Key (AppSKey) can be calculated. These IDs are also transmitted to the server as part of the JoinRequest command so that the NwkSKey and AppSKey can also be calculated and stored there. These keys are unique between a module and a server.

The following IDs must be configured for OTAA:

- **Application Identifier (AppEUI) –** This is a global application ID in IEEE EUI64 which uniquely identifies the application provider of the module.
- **End-device Identifier (DevEui) –** This is a global end-device ID in IEEE EUI64 which uniquely identifies the module/end-device.
- **Application Key (AppKey) –** This is an AES-128 application key specific to the module which is used to derive the session keys NwkSKey and AppSKey.

These values can be stored and read back from the EU module using the commands listed in Table 4. The values set using the **LORAMACSETOPTION** are not persisted over a power cycle.

Table 4: OTAA IDs

ID	Data Length (Bytes)	Write Command	Read Command
AppEui	8	at+cfgex 1010 “xxxx” (For firmware versions prior to 18.4.1.0 the AppEui Id is 1000.)	at+cfgex 1010?
		LORAMACSetOption(7, xxxx)	LoramacGetOption(5)
DevEui	8	N/A- Configured during production	ati 25
		at+cfgex 1011 “xxxx” (For firmware versions prior to 18.4.1.0 the DevEui Id is 1001.)	at+cfgex 1011?
		LORAMACSetOption(5, xxxx)	LoramacGetOption(7)
AppKey	16	at+cfgex 1012 “yyyy” (For firmware versions prior to 18.4.1.0 the AppKey Id is 1002.)	Write Only
		LORAMACSetOption(6,yyyy)	

Note: The **xxxx** above represents an 8-byte hexadecimal value and the **yyyy** a 16-byte hexadecimal value.

The IDs configured using the **at+cfgex** command only take effect after a module reset. This is not performed automatically as part of the command; it must be manually initiated.

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0600

IDs configured using the **LORAMACSetOption** API are only valid while the module is powered. The value is not persisted so is discarded when the module is powered down or reset.

Note: There are two options for configuring a permanent DevEui. The device is initially configured with a global DevEui during production. However, it is possible to override this value with a local definition using the **at+cfgex** option. When it comes to selecting a value, the code selects the local value first if it is available. If not, it reverts to the global value. The **ati 25** command only returns the global value and **at+cfgex 1011?** only returns the local value.

The LORAMACSetOpiton temporarily replaces the at+cfgex option.

On receipt of a **LORAMACJoin(LORAMAC_JOIN_BY_REQUEST)** command, the module first checks that all the required parameters are configured. If not, the join request is cancelled and an error message is returned indicating which parameter is missing.

Note: More details regarding all the *smart*BASIC commands referenced in this application note can be found in the RM1xx *smart*BASIC Extensions Guide, found in the documentation tab of the RM1xx Series product page.

If everything is working, the join request is transmitted to the gateway/server. If successful, an **EVLORAMACJOINED** event is passed to the user. This event is not received for at least five seconds after the transmission of the join request command due to the preprogrammed delays in the system. This is explained in the Data Reception section and the expected delays are defined in Table 6.

If the JoinRequest fails, the firmware enters a pre-programmed sequence, as explained above, where it transmits JoinRequests at reducing datarates, starting at DR5, until eventually a final JoinRequest is transmitted at DR0 which has the largest spreading factor and consequently the longest range. At the end of this sequence if no JoinAccept is received, the **EVLORAMACRXTIMEOUT** event is thrown. It is then up to the *smart*BASIC application as to what to do next.

Activation by Personalization

When using Activating by Personalization, the NwkSKey and the AppSkey must be configured on both the RM186 and any server with which it might communicate. By default, it is likely that the DevEui and AppEUi will also be set in the module as their values must be set on the network server, however these IDs are not used in the Join process.

- **Network Session Key (NwkSKey) –** Specific to the end device. Used by the module and gateway/server to calculate the checksum of all data messages. Also used to encrypt and decrypt the payload field of MAC only data messages.
- **Application Session Key (AppSKey) –** Specific to the end device. Used by the module and gateway/server to encrypt and decrypt the payload data. It may also be used to calculate the optional payload checksum.
- **End Device Address (DevAddr):** 32 bits (4 bytes) that identifies the module within the current network.

Table 5: Personalization IDs

ID	Data Length (Bytes)	Write Command	Read Command
NwkSKey	16	at+cfgex 1013 “xxx.xxxx” (For firmware versions prior to 18.4.1.0 the NwkSKey Id is 1003.)	Write Only

ID	Data Length (Bytes)	Write Command	Read Command
AppSKey	16	at+cfgex 1014 “xxx.xxx” (For firmware versions prior to 18.4.1.0 the AppSKey Id is 1004.)	Write Only
DevAddr	4	at+cfgex 1015 “xxxx” (For firmware versions prior to 18.4.1.0 the DevAddr Id is 1005.)	at+cfgex 1015?

Note: The **xxxx** above represents a hexadecimal value. For example:

at+cfgex 1013 "2b7e151628aed2a6abf7158809cf4f3c"

As with the OTAA, once a **LORAMACJoin(LORAMAC_JOIN_BY_PERSONALIZATION)** is received by the module, it checks that the above IDs are configured. If not, an error message is returned and the request is cancelled.

When you join a network using the personalization method, there is no handshaking required between the RM186 and the server. Both sides of the link should be configured with the same key values; the RM186 is assumed to have joined the network as soon as the command is sent. The module is ready to start transmitting data to the gateway immediately.

If using the Multitech gateway in Network Server mode, refer to the Conduit mLinux: LoRa Use with Third-Party Devices webpage for details on how to configure the gateway.

Note that when using this method of joining a network, there is no JoinAccept message; it is not possible for the network to immediately configure the module with the five additional channels allowed in the downlink packet. Additional channels may be added by another MAC command but this is dependent on how the network server is configured. The data throughput on modules using personalization may be less than on those using OTAA.

DATA TRANSMISSION

Transmitting data to the server is a simple task. Call the *smart*BASIC **LoramacTxData** command containing the data you wish to send to the server along with the port number and the confirm flag. Everything else is handled by the firmware.

A recent change to the RM1xx firmware has now made it possible to query the stack as to the maximum possible packet size using the **LORAMACQueryTxPossible** API, which returns the theoretical and actual maximum packet sizes. With this command it is now possible to control the data packets more efficiently. It is worth pointing out here that in the case of confirmed packets it is possible that although the packet is valid when you first send it, if it enters the resend phase, as the datarate reduces the packet size may become too large and the **EVLORAMACTXDRPAYLOADSIZEERROR** will be thrown.

As with OTAA, data packets are subject to the same duty cycle restrictions. The LoRaWAN stack searches all the enabled frequency channels to determine which, if any, have duty cycle available. If there are any available, it randomly selects one of those channels and transmits the packet to the server. If not, the module waits for the first frequency channel to become available and then transmits the packet.

There are two options as to when to load a new data packet into the stack once the previous uplink/downlink sequence is complete.

You can simply just load the packet. If there is duty cycle available it is transmitted. If not, the stack waits until it does become available and then sends it. Once a packet has been loaded, you can check when it is transmitted

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0600

using the **LORAMACGetOption(LORAMAC_OPT_NEXT_TX,var$)** API. This command only returns a useful value once a packet is loaded into the stack.

Once a packet is loaded it cannot be deleted or overwritten. This packet is the next packet transmitted once there is available duty cycle.

The other option is to wait until there is duty cycle available before loading the packet into the stack. There are two options here. You can wait for the **EVLORAMACNEXTTX** event which indicates that there is available duty cycle. Or you can poll **LORAMACGetOption(LORAMAC_OPT_NEXTTX_TIME,var$).** When it returns 0, a packet is sent immediately upon loading.

There are several ways the successful completion of an uplink/downlink sequence can be determined using the LoRa events.

You should always receive an **EVLORAMACTXCOMPLETE** event, provided nothing has gone wrong with the transmission or reception of data (such as you received an **EVLORAMACTXDONE** event) and nothing has gone wrong with any received packet. If there is an error in the received packet the **EVLORAMACTXCOMPLETE** is not thrown.

The only exception is when a confirmed transmit packet is rejected by the server, but the server has a downlink packet to transmit. The downlink packet would still be transmitted so in this case you would receive the **EVLORAMACRXCOMPLETE** event but not the **EVLORAMACTXCOMPLETE** event as the uplink would not have been acknowledged.

If there is a successful downlink packet you should also receive an **EVLORAMACRXCOMPLETE** event. This confirms an acknowledgment to a confirmed uplink packet. If that packet also contains data there is also an **EVLORAMACRXDATA** event.

To make the process simpler, a new event was created – **EVLORAMACSEQUENCECOMPLETE**. Internally the firmware monitors what is supposed to happen and which event should end the uplink/downlink sequence. This event is then thrown along with a variable indicating which event was the actual terminating event. All that is now required is to monitor the **EVLORAMACSEQUENCECOMPLETE** and act on that.

Examples of when the events are thrown are shown below. The waveforms are obtained using the **LoramasSetDebug** API. The blue waveform represents when the module is transmitting and the red waveform represents when the module is in receive mode.

In Figure 1 the module has been programmed to transmit a confirmed packet. Once the module stops transmitting the **EVLORAMACTXDONE** event is thrown. One second later the first receive window opens. This window is open for a minimum of eight preamble characters. If the module doesn't receive these preamble characters from the gateway, it closes. In this case, it does receive these characters and stays open. At the end of the window, once all the downlink packet has been received, the window closes and the module throws the following:

- **EVLORAMACTXCOMPLETE** event – Because it has successfully transmitted a packet
- **EVLORAMACRXCOMPLETE** event – because the downlink packet was received correctly
- **EVLORAMACSEQUENCECOMPLETE** event – Returned with a numeric value of 2

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0600

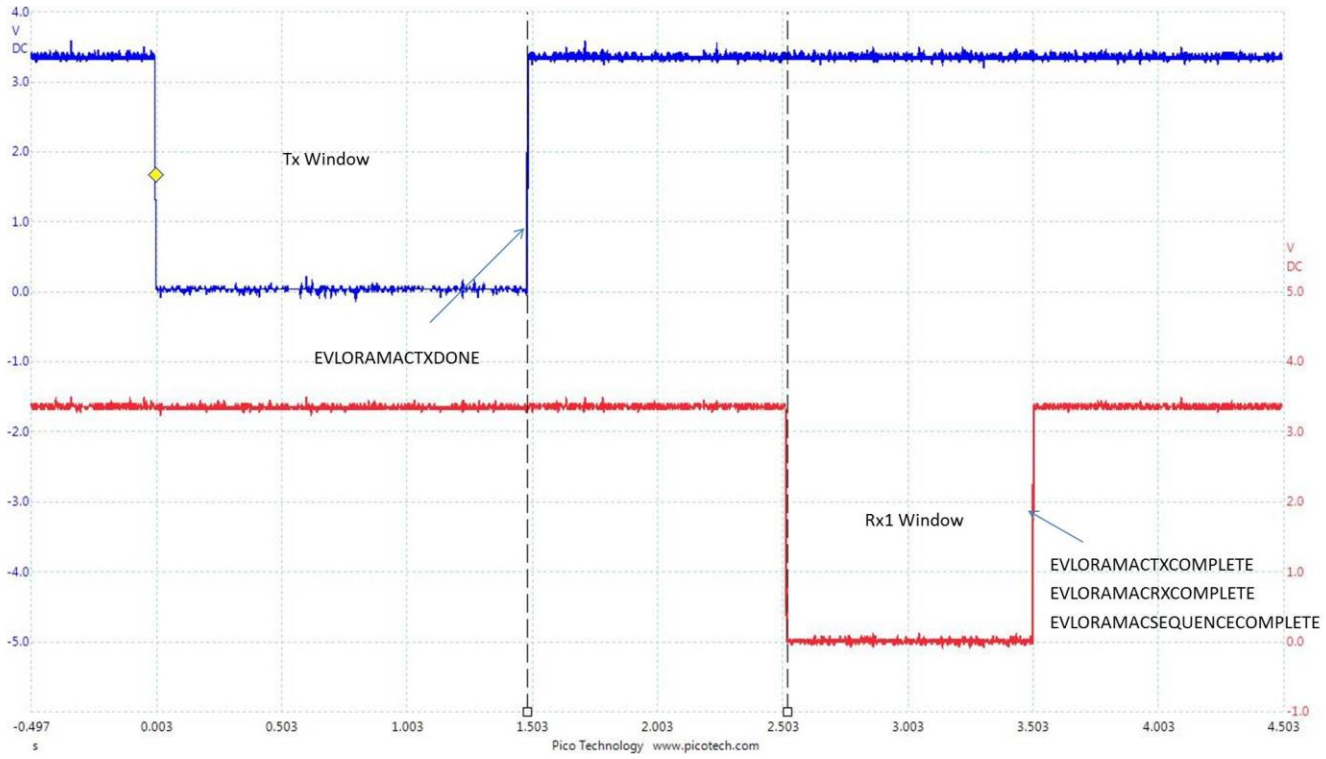


Figure 1: Confirmed packet

Figure 2 shows an example of an unconfirmed packet transmitted to the gateway. The transmit trace is the same as before. However, because it was an unconfirmed uplink there is no downlink packet. The receive windows only open for the minimum time causing the **EVLORAMACNOSYNC** events to be thrown. In this example, the numeric value returned with the **EVLORAMACSEQUENCECOMPLETE** event would be 1.

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0600

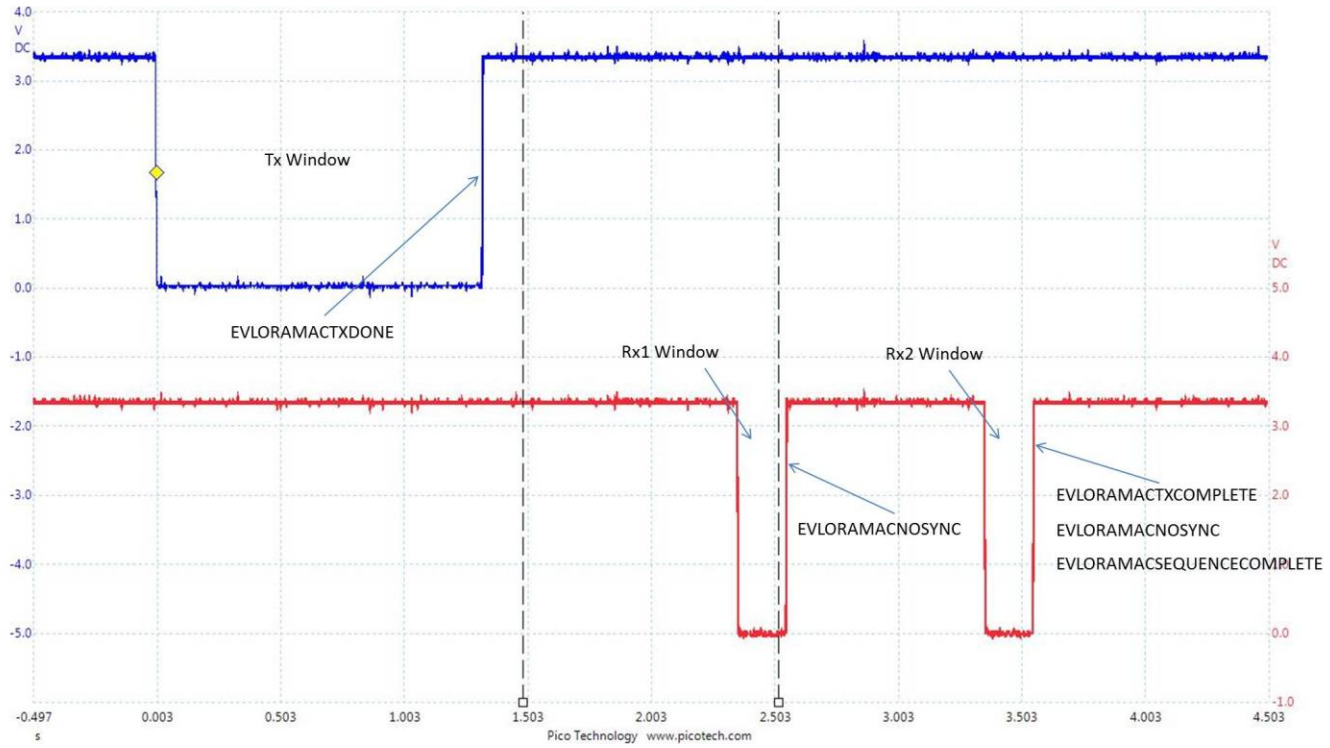


Figure 2: Unconfirmed data packet

As mentioned in an earlier section, if the transmission fails, the confirmed/not confirmed option determines what happens next. If the confirmed option is selected and the confirmation is not received, the module attempts to retransmit the packet. By default, the module attempts to send a specific packet eight times (the initial attempt and seven retries). The number of retries can be configured using the **LORAMACGetOption(LORAMAC_OPT_MAX_RETRIES,var$)** command. The number of retries cannot exceed eight. If the number of retries is reached without a receiving a response, an EVLORAMACRXTIMEOUT event is thrown.

During this retransmit process the LoRaWAN stack automatically decrements the data rate after every two failed transmissions. So, depending on the starting data rate, the data rate at the end of this process could be much lower than at the beginning, potentially causing this process to take longer than expected. The module tends towards decreasing the data rate as this has the effect of increasing the range of the module. Reducing the datarate could have the effect of now making the packet too large for transmission causing a **EVLORAMACTXDRPAYLOADSIZEERROR** event to be thrown.

If an **EVLORAMACRXTIMEOUT** event is received, it is up to the user how to proceed. You can either try to join the network again or try sending another packet.

With the RM186, once the uplink/downlink sequence is complete, the module automatically calculates how long until the next uplink packet can be transmitted. When that time has elapsed, the **EVLORAMACNEXTX** event is thrown.

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0600

DATA RECEPTION

After each transmitted uplink packet on a Class A LoRaWAN device, the module must listen for a downlink packet, which may or may not be sent from the gateway There are two possible receive windows in which this downlink packet could be transmitted, illustrated in Figure 3.

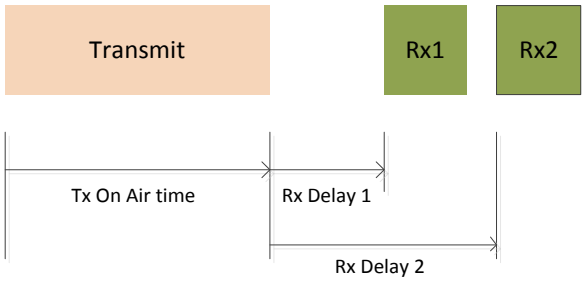


Figure 3: Receive window timings

The actual length of the delays (Table 6) is dependent on whether the RM186 is transmitting a join request or a normal data packet.

Table 6: Receive delay times

Packet Type	Rx Delay 1	Rx Delay 2
Join Request	5 seconds	6 seconds
Data Packet	1 second	2 seconds

In the first receive window, the gateway transmits on the same frequency as the uplink message. In the second receive window, the gateway transmits on the default 869.525 MHz at data rate DR0.

If the downlink packet is transmitted in the first window and is successfully received by the module, then the second window is not required and does not open.

If the downlink packet is not received during the first Receive window, then the second Receive window is opened and the module listens on 869.525 MHz. A likely scenario is that the gateway ran into duty cycle issues on the uplink frequency channel and had no time remaining for that band.

As shown in Figure 3, all the timings are taken from the end of the uplink transmission. This is the **EVLORAMACTXDONE** event. At the designated interval, the RM186 turns on the receiver for a short period and listens for a sync message. If the signal is not detected, the receiver is switched off and the RM186 then waits for the second window to open and repeat the process.

If the sync message is detected in any of the receive windows, the RM186 receiver remains switched on until the full downlink packet is received. In this way the RM186 saves power by only having the receiver switched on for the minimum amount of time.

The gateway only transmits the downlink on one of the receive windows. It has no knowledge if something has gone wrong with the reception so it does not resend the downlink in the 2nd window. However, it is possible for the gateway to make the downlink a confirmed packet so the next uplink must contain an acknowledgment. This is all handled internally in the stack.

Note: If the sync message is transmitted outside this initial small receive window, the downlink packet is missed. The LoRa gateway has no knowledge of this fact. As far as the gateway is concerned, the packet was sent. It is not retransmitted in the second window.

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0600

Again, the uplink packet option (confirmed or not confirmed) determines the RM186's response to this. If confirmed is selected, then the same uplink packet is resent, following the procedure outlined above.

Receive timings are hardcoded. There are no configuration options available to the user to modify these timings. The RM186 timings meet those of the LoRaWAN specification. It is the gateway's responsibility to send the downlink packet at the correct time.

Link Check

A module can transmit a Link Check request to the server. This command contains no payload.

The Link Check response from the server contains an indication of the signal strength of the last Link Check request received by the server and the number of gateways that have received that request.

Controlling the Personalization Sequence number

The LoraWan specification states that, when a module has joined a network using the personalization option, it must keep track of the uplink and downlink counter values over a reset. The reason for this is that a module is deemed to have "Joined" a network when the keys are loaded into both the module and the network server. There are no handshaking messages. So when the module is powered down it is still, in effect, joined to that network.

For both uplink and downlink packets the main requirement is that the counter value of an incoming packet must be less than 16384 greater than that of the previous received packet. If the value is greater than 16384 more than the previous packet the incoming packet is rejected. No indication is sent back to the transmitting device on why the incoming packet was rejected.

Also worth noting is that if the incoming packet counter value is less than that of the previous packet, for example either the module or server has been reset, this is likely to result in a negative difference which will probably equate to a large positive number greater than 16384 and so will also be rejected.

It is important to note that the incoming counter value does not necessarily have to be just one more than the previous value. Provided the difference is less than 16384, the incoming packet will be accepted.

To support this requirement the RM1xx now stores uplink and downlink counter values in the internal serial eeprom that will be used to initialise the internal counters on boot up, before the first uplink packet is transmitted. However, to ensure we're not writing to the serial eeprom every time we transmit and receive packets the module uses an incremental step to determine when the values need to be updated.

This incremental value defaults to 256 but it can be modified in Flash using the **at+cfg 1003** command or temporarily using the **LoramacSetOption(LORAMAC_OPT_SEQUENCE_INCREMENT, ..)** smartBASIC command. In both cases the value must lie between 1 and 256.

Looking at the uplink counter first and starting with the scenario where everything has been reset, the first uplink counter value will be 0. If the incremental step is set to the default value of 256, this means that the uplink counter stored in the serial eeprom will be 256.

Every time an uplink packet is transmitted the counter value of the uplink packet will be increased by 1 byte the stack. When it reaches 256 then the value in the serial eeprom will be incremented by 256 up to 512. On subsequent uplink packets the counter value will continue to increase by one every packet.

If, for whatever reason, the module then resets it will lose this running counter value. However when it reboots it will read in the value stored in the serial eeprom and so start counting from, following from the previous paragraph, 512. This, in turn, will cause the value in the serial eeprom to update to 768.

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0600

In the scenario where a module will only send a single packet before going to deep sleep (which requires a reboot to wake up from), it the incremental step is 256 then the first 4 uplink packet sequence numbers will be 0, 256, 512 and 768.

The downlink counter is handled slightly differently. It uses the same incremental step value but the value stored in the eeprom is only updated when the actual counter value is the incremental step greater than the stored value. So, again using the default value of 256, for downlink counter values of 0-255 the value stored in the eeprom will be 0. When the downlink counter reaches 256 the value in the eeprom is updated to 256.

So if the module resets and the actual downlink counter lies between 0 and 255, it will initialise with 0. If the actual value is greater than 256 to 511 it will be 256. This ensures that the local value is close enough to the next incoming value to not cause problems.

Also supporting these counter operations are the **LoraMacSetOption/LoramacGetOption** commands, **LORAMAC_OPT_EEPROM_UPCOUNTER** and **LORAMAC_OPT_EEPROM_DOWNCOUNTER**. These can be used to set or read the stored sequence number value stored in the serial eeprom. This has the effect of setting the sequence number to a value of your choosing after a reset.

The actual real values being passed between the module and the server can be retrieved using the **LORAMAC_OPT_DOWNLINK_COUNTER** and **LORAMAC_OPT_UPLINK_COUNTER** ids. These are read only commands.

REFERENCES

- Lora Alliance - LoRaWAN Specification – The current version is available from the LoRa Alliance website.
- LoRaWAN Regional Parameters – The current version is available from the LoRa Alliance website.
- User Guide – RM1xx Series *smart*BASIC LoRa Extensions (documentation tab of RM1xx product page)
- User Guide - *smart*BASIC Core Functionality (documentation tab of RM1xx product page)
- Application Note – Connecting to Multitech Conduit Gateway (documentation tab of RM1xx product page)
- http://www.multitech.net/developer/software/lora/conduit-mlinux-lora-communication/conduit-mlinux-lora-use-third-party-devices/

REVISION HISTORY

Version	Date	Notes	Contributor	Approver
1.0	20 May 2016	Initial Release	Colin Anderson	Jonathan Kaye
1.1	10 Oct 2016	Updates to several ID values	Colin Anderson	Jonathan Kaye
1.2	11 Oct 2017	Updated for version 100.6.1.0 & 110.6.1.0	Colin Anderson	Jonathan Kaye
1.3	16 Nov 2017	Fixed typo in *default Data Rate configuration* section	Colin Anderson	Jonathan Kaye