IBM Long Range Signaling and Control

IBM LoRaWAN Modem

(Feather M0 Edition)

Version 2.1
11 - October - 2019

IBM LoRaWAN Modem Product Information

LoRaWAN Modem is developed and marketed by the IBM Zurich Research Laboratory (IBM Research GmbH), 8803 Rüschlikon, Switzerland. For additional information please contact: lrsc@zurich.ibm.com.

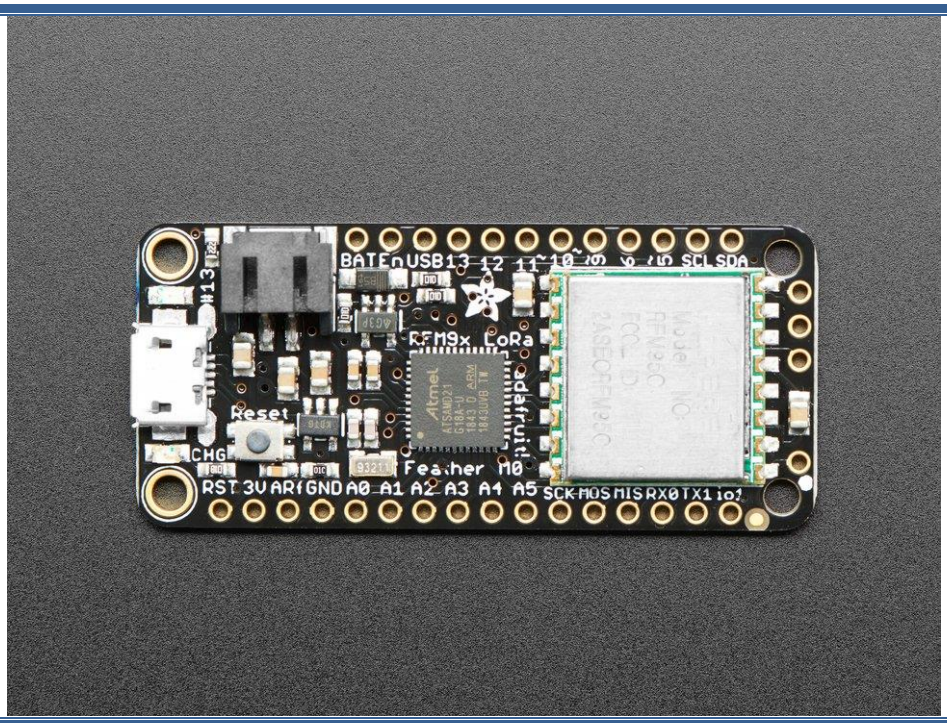
Table of Contents

1. Introduction

The IBM LoRaWAN Modem provides access to LoRaWAN networks via high-level commands exchanged over a serial interface. All kinds of devices can be easily enabled to participate in LoRaWAN networks just by connecting the modem via a serial link and sending a few commands. The modem offers full functionality over a simple interface and handles all details of the LoRaWAN protocol internally. It can be personalized and configured with the specific network parameters.

The modem firmware is based on the Basic MAC library forked from IBM LoRaWAN C-library (LMiC). This edition is running on the Adafruit Feather M0 with RFM9x LoRa module, featuring a ATSAMD21G18 ARM Cortex M0 MCU and the Semtech SX1272 LoRa™ radio. The firmware can be run directly on the Adafruit Feather M0 with RFM9x LoRa module without any additional HW modification except optional antenna wire (Figure 1).

Figure 1. *Adafruit Feather M0 with RFM9x LoRa module*



2. Modem Interface

2.1 Connection

The modem is connected to the end device via a USB interface emulating serial COM port with the communication settings 115200bps, 8/N/1, using half-duplex mode.

2.2 LED Indicators

On board red LED is used to indicate internal modem states.

Activation States:

Activation State	Blink period [s]
Not activated	1
Activated	5
Joining	0.1

Where LED ON duration is short battery saving time 5ms.

Hard Fault state:

LED blinking in 1s period with ON:OFF ratio 9:1

Data transmitted:

LED ON for 0.5s

2.3 Messages

The modem processes all radio messages and LoRaWAN protocol states internally and can be driven by the end device over the serial link. It recognizes a set of commands to configure and query parameters and to initiate and report data exchanges with the network.

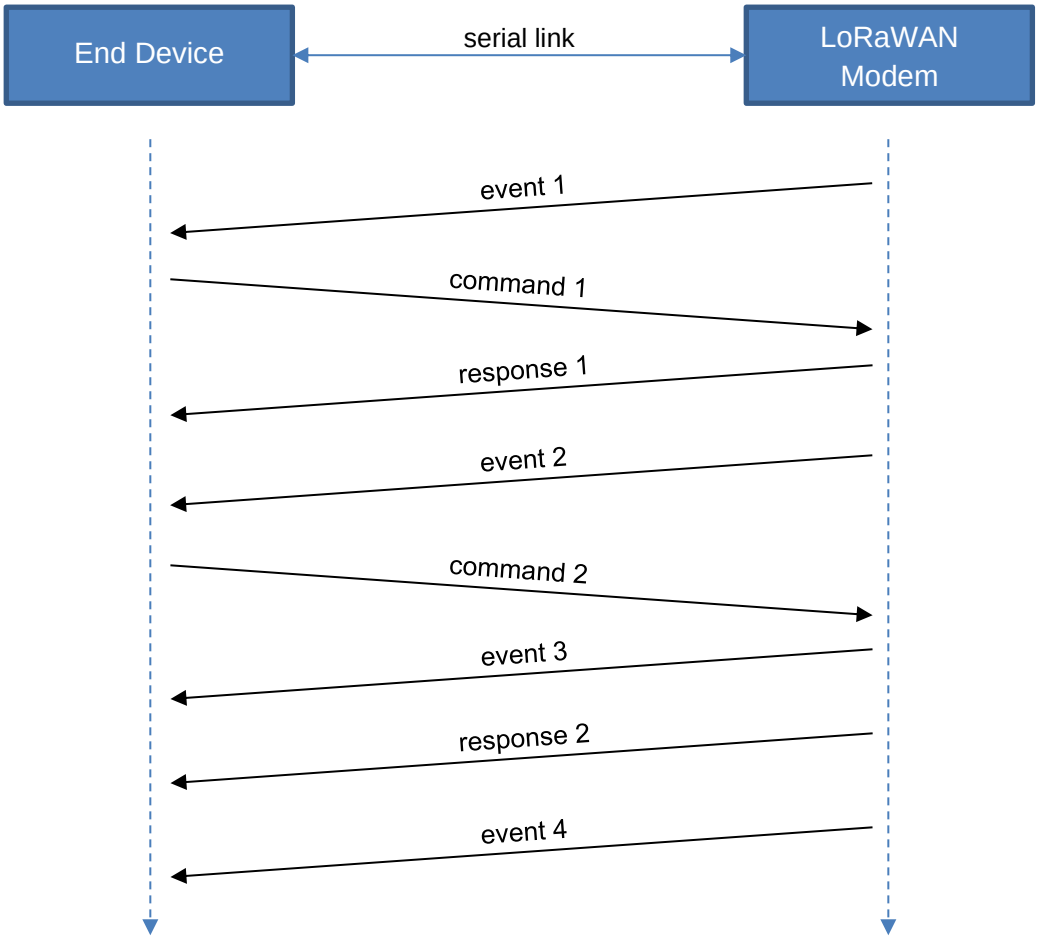
All commands are answered by the modem with a corresponding response message. The end device must not issue a new command before it has received the response to the previous command.

In addition to the response messages the modem can generate event messages. These event messages can inform the end device about certain state changes inside the modem triggered by the protocol. Given the nature of the LoRaWAN protocol these state changes and corresponding events can occur at any time. I.e. the end device application must be always prepared to accept event messages. This is even the case when a command/response exchange is in progress. If the generation of the event overlaps with the sending of a command, the event message could be received by the end device **before** the expected response message!

The types of event messages to be reported by the modem can be configured via an event mask (see section 4.11). Possible message flows are shown in Figure 2.

Currently the modem supports an ASCII AT command set (see section 4), inspired by the command set of Hayes data modems. A binary command set is planned.

Figure 2. *Modem Message Flow*



Exchange of command, response, and event messages.

2.4 Power Consumption

Modem supports low power mode allowing operation from battery consuming less then 100uA in idle state. Low power is activated when powered from battery, it means not connected to terminal using USB interface.

3. Modem State

The modem has a persistent memory to store its internal state. The stored state is continuously updated, so the modem can continue operation after a power cycle without reconfiguration. Most importantly the activation state (session state) is stored and updated. This includes session keys and frame sequence counters.

3.1 Activation

To be able to exchange data with the network server the modem needs to be activated. A valid activation state requires session parameters.

3.1.1 Session Parameters

The session parameters consist of:

- Network ID (32 bit)
- Short device address (32 bit)
- Network session key (128 bit)
- Application session key (128 bit)

These parameters can be obtained by three ways:

1. embedded in the modem firmware (personalized activation, see section 3.2.1)
2. configured by a modem command (personalized activation, see section 4.5.1)
3. established via the JOIN command (over-the-air activation, see sections 3.1.2 and 4.7)

Once obtained the session parameters are stored and updated in the persistent storage. Whenever new session parameters are set, the upstream and downstream sequence counters are reset to zero.

3.1.2 Join Parameters

If the modem is to be activated via over-the-air-activation (JOIN), the following join parameters are required:

- Device EUI (64 bit)
- Application EUI (64 bit)
- Device Key (128 bit)

The join parameters can be obtained by two ways:

1. embedded in the modem firmware (see section 3.2.2)
2. configured by a modem command (see section 4.6.1)

Using the join command (see section 4.7) a new session is established over-the-air and the obtained session parameters are stored and updated in the persistent storage.

3.2 Firmware Personalization

In some cases it is desirable to have the modem firmware preconfigured with the specific network parameters, so the modem is operational without any configuration commands. Therefore the parameters can be patched into the HEX file containing the firmware. Special patterns have been embedded in the firmware to identify the locations of the parameter blocks to be patched.

3.2.1 Session Parameters

The block of session parameters can be patched into the firmware file where the following 40-byte HEX-pattern is found:

```
426d6633717561434a77564b5552575752456547 4b746d3070714c4430596872356370506b503673
```

The layout of the patched session parameters is as follows:

Parameter	Size in bytes	Remark
Network ID	4	least-significant-byte-first
Short device address	4	least-significant-byte-first
Network session key	16	
Application session key	16	

3.2.2 Join Parameters

The block of join parameters can be patched into the firmware file where the following 32-byte HEX-pattern is found:

```
6730434d7734397252626176364877514e303131356734324f706d76546e3771
```

The layout of the patched join parameters is as follows:

Parameter	Size in bytes	Remark
Device EUI	8	least-significant-byte-first
Application EUI	8	least-significant-byte-first
Device Key	16	

4. AT Command Set

Modem commands, responses and events are encoded as ASCII strings terminated by a carriage-return character <CR>. This way the modem can be directly accessed with the terminal application of your choice and the commands can be typed and read in a human-readable form.

All commmands are prefixed with the characters "AT", followed by comma-separated parameters, and terminated by <CR>. The interpretation of the commands is case-insensitive. All commands will be answered by the modem with "OK" and optional parameters and <CR>, or "ERROR<CR>". As mentioned in section 2.3, intermittent events can be generated by the modem before the response is sent. Event names are prefixed with "EV_" and are followed by optional comma-separated parameters and a trailing <CR>.

All available commands with their responses and possible events are described in detail in the following sub sections.

4.1 NOP Command

An empty command can be used to test the communication between the modem and the end device. It performs no operation.

Command	AT
Response	OK

Example:

```
→ AT
← OK
```

4.2 Version Command

This command can be used to query the firmware version of the modem. It will return a fixed-length string containing the major and minor version number and the compile time of the modem firmware.

Command	ATV?
Response	OK,*<version string>*

Example:

```
→ ATV?
← OK,VERSION 1.2 (May  8 2015 16:32:38)
```

4.3 Reset Command

This command resets the internal state engine and reloads session state from persistent memory.

Command	ATZ
Response	OK

Example:

```
→ ATZ
← OK
```

4.4 Factory Reset Command

This command resets the persistent memory to factory state. All parameters will be reverted to the values built into the firmware or cleared. Manually configured parameters and progressed session state will be discarded.

Command	AT&F
Response	OK

Example:

```
→ AT&F
← OK
```

4.5 Session Parameter Commands

The following two commands allow to set and query the session parameters.

4.5.1 Set Session Parameter Command

This command allows to directly set the session parameters. The new session parameters will be written to persistent memory and sequence counters will be reset to zero.

Command	ATS=*<network id>,<device address>,<network session key>,<application session key>*
Response	OK

Example:

```
→ ATS=00000002,05A49FEC,00112233445566778899AABBCCDDEEFF,00112233445566778899AABBCCDDEEFF
← OK
```

4.5.2 Query Session Parameter Command

This command returns the current session parameters. If the modem is not activated, i.e. no session exists, ERROR will be returned. **Note:** The session keys are not returned by the modem!

Command	ATS?
Response	OK, *<network id>,<device address>,<up sequence counter>,<down sequence counter>*

Example:

```
→ ATS?
← OK,00000002,05A49FEC,00000004,00000003
```

4.6 Join Parameter Commands

The following commands allow to set and query the join parameters which will be used by the JOIN command (section 4.7) for over-the-air activation.

4.6.1 Set Join Parameter Command

This command allows to directly set the join parameters.

Command	ATJ=*<device EUI>,<application EUI>,<device key>*
Response	OK

Example:

```
→ ATJ=FFFFFFFFFFFFFF00,DEDEAAAA0000001A,AA55555555555555AAAAAAAAAAAAAA
← OK
```

4.6.2 Query Join Parameter Command

This command returns the join parameters. If no join parameters are set, ERROR will be returned.
Note: The device key is not returned by the modem!

Command	ATJ?
Response	OK,*<device EUI>,<application EUI>*

Example:

```
→ ATJ?
← OK,FFFFFFFFFFFFFF00,DEDEAAAA0000001A
```

4.7 Join Command

This command triggers the over-the-air activation using the configured join parameters to establish a new session. If no join parameters are configured the command will return ERROR.

Command	ATJ
Response	OK
Events	EV_JOINING EV_JOINED

The command immediately triggers the JOINING event and the session LED starts blinking. When the join procedure succeeded a JOINED event is generated and the session LED is turned on. The newly

established session parameters will be stored persistently and the sequence counters will be reset to zero.

Example:

```
→ ATJ
← OK
← EV_JOINING
← EV_JOINED
```

4.8 Transmit Command

This command is used to send upstream data to the network server. If the modem is not activated and join parameters are set, the modem will implicitely join and establish a new session. If the modem is not activated and no join parameters are set, `ERROR` will be retuned.

Command	ATT*<confirmed>,<port>[,<data>]*
Response	OK
Events	EV_TXCOMPLETE,*<flags>[,<port>[,<data>]]*

The data is addressed to a specific port (`01-FF`) and can be requested to be confirmed (`0` or `1`). After the data is sent by the modem it will check for downstream frames sent by the server. These frames could contain protocol information (like ACK or NACK), or application data. In any case an `EV_TXCOMPLETE` event will be generated to signal transmission and optional reception (see section 4.13 for description of event messages). **Note:** The data will only be sent at the point in time allowed by the modem's duty cycle.

Example:

```
→ ATT0,FF,1122334455            (send data unconfirmed to port 255)
← OK
← EV_TXCOMPLETE,00              (data sent, nothing received)

→ ATT1,03,112233                (send data confirmed to port 3)
← OK
← EV_TXCOMPLETE,A2              (data sent, ACK received in second window)

→ ATT0,04,112233                (send data unconfirmed to port 4)
← OK
← EV_TXCOMPLETE,02,0A,C0FFEE    (data sent, data received in second window on port 10)
```

4.9 Ping Mode Command

This command enables ping mode and prepares the modem for slotted reception of unsolicited downstream data (class B network). The modem will immediately start scanning for a beacon, and if found, it will deliver an `EV_BEACON_FOUND` event. This command requires a valid session. If no session exists, `ERROR` will be returned.

Command	ATP<*interval exp*>
Response	OK
Events	EV_BEACON_FOUND EV_RXCOMPLETE,<*flags*>[,<*port*>[,<*data*>]]

The ping interval is specified as an exponent (0-7) and is defined as $2^{<\text{interval exp}>}$ seconds. Whenever data is received in a ping slot window, an EV_RXCOMPLETE event is generated. See section 4.13 for a detailed description of event messages. **Note:** The network server is only notified of the ping mode and the interval with the next upstream frame sent by the modem!

Example:

```
→ ATP2                                  (set 4sec ping interval and scan for beacon)
← OK
← EV_BEACON_FOUND

→ ATT0,01,AABBCC                        (send data, notify server of ping interval)
← OK
← EV_TXCOMPLETE,00

← EV_RXCOMPLETE,0P,01,112233            (data received in ping window on port 1)
← EV_RXCOMPLETE,0P,01,4455667788        (data received in ping window on port 1)
```

4.10 Alarm Timer Command

This command sets an alarm timer to the specified number of seconds (variable-length HEX). When the timer expires an EV_ALARM event is generated. It can be used by the end device to periodically wake up or schedule actions at a specified time.

Command	ATA<*seconds*>
Response	OK
Events	EV_ALARM

Example:

```
→ ATA1E                                 (request ALARM event in 30 seconds)
← OK
← EV_ALARM
```

4.11 Event Mask Commands

The following commands allow to modify and query the event types which will be reported by the modem. Events not enabled in the event mask will be silently dropped and not reported by the modem.

4.11.1 Query Event Mask Command

This command returns the current event mask. By default all events are enabled.

Command	ATE?
Response	OK,*<event mask>*

Event mask can be ALL, or NONE, or a list of event names separated by the "|" character.

Example:

```
→ ATE?
← OK,ALL

→ ATE?
← OK,JOINED|TXCOMPLETE|RXCOMPLETE
```

4.11.2 Set Event Mask Command

This command directly sets the event mask to the specified event types. Event names can be abbreviated and all events matching the prefix will be included. Special masks ALL and NONE are recognized by this command.

Command	ATE=*<event mask>*
Response	OK

Example:

```
→ ATE=ALL                              (select all events)
← OK

→ ATE=JOIN|TX|RX                       (select events beginning with JOIN* TX* RX*)
← OK
→ ATE?
← OK,JOINING|JOINED|JOIN_FAILED|TXCOMPLETE|RXCOMPLETE
```

4.11.3 Add Event Mask Command

This command adds events to the current event mask.

Command	ATE+*<event mask>*
Response	OK

Example:

```
→ ATE+JOIN             (add JOINING, JOINED, JOIN_FAILED to the current event mask)
← OK
```

4.11.4 Remove Event Mask Command

This command removes events from the current event mask.

Command	ATE-<event mask>
Response	OK

Example:

```
→ ATE-RX|TX              (remove RXCOMPLETE, TXCOMPLETE from the current event mask)
← OK
```

4.12 RF band Parameter Commands

The following commands allow to set and query the RF band parameter which will be used by the radio module.

List of available RF bands:

RF band	Description
01	eu868
02	as923
03	us915
04	au915
05	cn470

4.12.1 Set RF band Parameter Command

This command allows to directly set the RF band parameter. If RF band parameter is not available, the command will return ERROR.

Command	ATR=<RF band>
Response	OK

Example:

```
→ ATR=03                                (select us915 RF band)
← OK
```

4.12.2 Query RF band Parameter Command

This command returns the RF band parameter.

Command	ATR?
Response	OK,<RF band>

Example:

```
→ ATR?
← OK,03
```

4.13 Events

The following events can be generated by the modem if they are enabled in the event mask.

- **EV_JOINING**
  The modem has started joining the network.
- **EV_JOINED**
  The modem has successfully joined the network and is now ready for data exchanges.
- **EV_JOIN_FAILED**
  The modem could not join the network (after retrying).
- **EV_REJOIN_FAILED**
  The modem did not join a new network but is still connected to the old network.
- **EV_TXCOMPLETE**
  The data has been sent, and eventually downstream data has been received in return. If confirmation was requested, the acknowledgement has been received.
- **EV_RXCOMPLETE**
  Downstream data has been received.
- **EV_SCAN_TIMEOUT**
  No beacon was received within the beacon interval.
- **EV_BEACON_FOUND**
  The first beacon has been received.
- **EV_BEACON_TRACKED**
  The next beacon has been received at the expected time.
- **EV_BEACON_MISSED**
  No beacon was received at the expected time.
- **EV_LOST_TSYNC**
  Beacon was missed repeatedly and time synchronization has been lost.
- **EV_RESET**
  Session reset due to rollover of sequence counters. If configured, the join parameters will be used to automatically rejoin the network.
- **EV_LINK_DEAD**
  No confirmation has been received from the network server for an extended period of time. Transmissions are still possible but their reception is uncertain.

Most events don't have return parameters and will be reported only with the event name. Only the two events `EV_TXCOMPLETE` and `EV_RXCOMPLETE` do have return parameters:

- `EV_TXCOMPLETE`,*<recv flags>[,<port>[,<data>]]*
- `EV_RXCOMPLETE`,*<recv flags>[,<port>[,<data>]]*

Both event messages are always followed by flags to indicate the reception state. Optionally these events are appended with port information and the received application data.

The reception flags are coded as two ASCII digits <*X*><*Y*> with the following meaning:

Digit Value	Description
<*X*> = `0`	No information
<*X*> = `A`	Frame acknowledged, ACK
<*X*> = `N`	Frame not acknowledged, NACK
<*Y*> = `0`	No frame received
<*Y*> = `1`	Frame received in down window 1

<*Y*> = 2	Frame received in down window 2
<*Y*> = P	Frame received in ping slot

5. Release History

Version and date	Description
V 1.0 February 2015	Initial version.
V 1.1 March 2015	Minor internal fixes. Document reformatting.
V 1.2 May 2015	Rebuilt using LMiC v1.5.
V 2.0 October 2019	IMST WiMOD module deprecated Adafruit Feather M0 with RFM9x LoRa module introduced instead Rebuid using LMiC v2.1. (LoRa Basics™ MAC)
V 2.1 October 2019	Added RF band selection support, selectable using ATR command