
# Embit Binary Interface Overview

**embit** s.r.l.

# Document information

## Versions & Revisions

| Revision | Date | Author | Comments |
|---|---|---|---|
| 1.0 | 01/01/2012 | A. Sala | Preliminary |
| 1.1 | 14/12/2012 | C. Biagi | Minor Fixes |
| 1.2 | 12/03/2013 | F. Montorsi | Minor Corrections; added hyperlinks |
| 1.3 | 05/04/2013 | C. Biagi | Minor Fixes |
| 1.4 | 19/04/2013 | A. Sala | Documented ATI command |
| 1.5 | 16/05/2013 | C. Biagi | Modified bootloader section |
| 1.6 | 31/07/2013 | A. Sala | Introduced the peripheral commands |
| 1.7 | 26/02/2014 | F. Montorsi | Reorganized and added figures |
| 1.8 | 05/03/2014 | F. Montorsi | Added check_packet() example func. |
| 1.9 | 14/03/2014 | C. Biagi | Minor fix |
| 2.0 | 05/05/2014 | F. Montorsi | Add list of EBI features |

## References

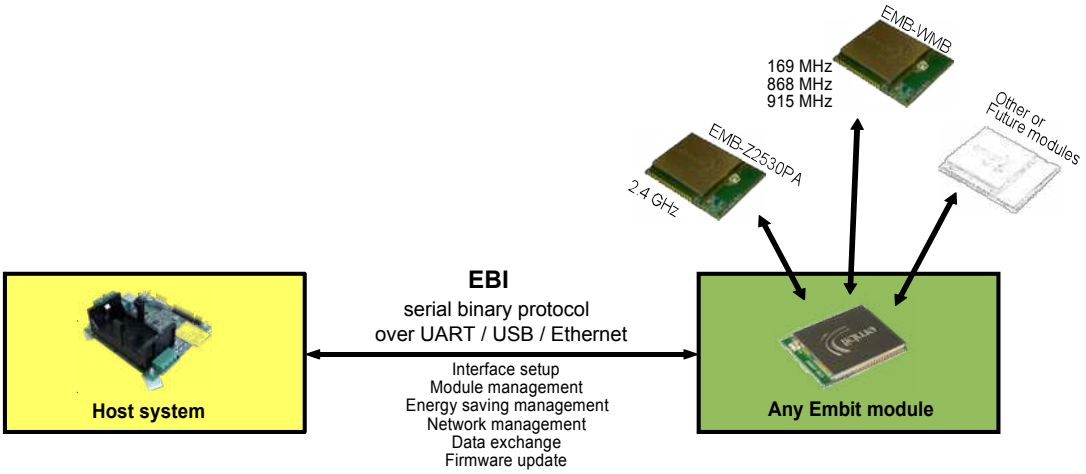
| Ref | Version | Date | Author | Title |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

# Index

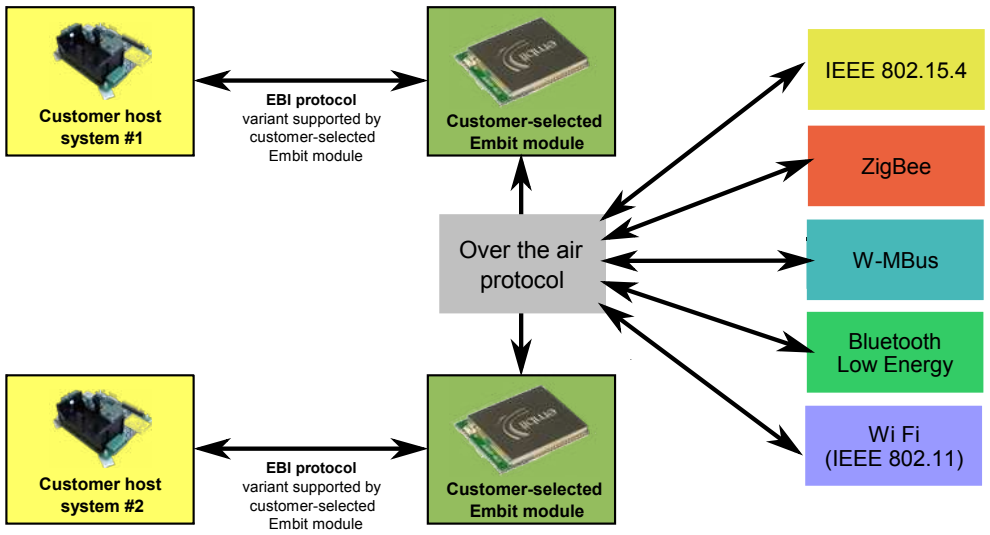
# 1 Introduction

This document describes the **Embit Binary Interface (EBI)** protocol available on the wireless OEM modules from Embit.

EBI can be described as a "*user-friendly serial protocol*" that allows to easily set-up a wireless network employing simple "*AT-like*" commands (over a UART interface) using Embit wireless modules:




A firmware application implementing the EBI protocol is available for all Embit modules, so that any Embit wireless module can be employed as a "flexible modem" using simple AT-like commands. The EBI protocol also accounts for the fact that different Embit wireless modules support different over-the-air protocols (e.g., some of them support only Wireless M-Bus, others support both IEEE 802.15.4 and ZigBee, etc):



In particular, the EBI protocol supports several commands that *abstract* the features of over-the-air protocols. For this reason, the majority of EBI commands are common to all Embit modules; a portion of the EBI commands are available only on some specific Embit wireless modules (see Section 2.4). Due to this, EBI documentation is divided in several documents:

- **"EBI Overview"**: this document; applies to all Embit modules;
- **"EBI W-MBUS-specific Documentation"** for all Embit modules supporting the W-MBus over-the-air protocol;
- **"EBI 802.15.4-specific Documentation"** for all Embit modules supporting the IEEE 802.15.4 over-the-air protocol;
- **"EBI ZigBee-specific Documentation"** for all Embit modules supporting the ZigBee over-the-air protocol.
- **"EBI Bootloader Guide"**; applies to all Embit modules supporting a bootloader.

The binary commands that are defined in the EBI protocol, described in details in the documents mentioned above, make easy to perform all the tasks required to setup a wireless communication:

- network formation (depending on the over-the-air protocol employed this includes association, security management, etc);
- management of the radio channel, RF output power, over-the-air data rate (for EBI-WMBUS), etc;
- transmission and reception of variable-length binary packets;
- radio scans (useful to find the less-crowded radio channels that allow for higher signal-to-noise ratios);
- device information retrieval (to identify the Embit radio module);
- enable/disable of power-saving modes (very useful for low-power wireless battery-operated);
- firmware update over serial port.

The next chapter provides an overview of the packet format used for all EBI commands.

# 2 EBI Binary Commands Overview

This chapter introduces the format of the binary commands and some general aspects of the EBI protocol. A complete description and payload specification for each EBI command is instead provided in the documents mentioned in Chapter 1:

- “EBI WMBus-specific Documentation”;
- “EBI 802.15.4-specific Documentation”;
- “EBI ZigBee-specific Documentation”.

However please note that a careful reading of this chapter is strongly suggested as first step.

## 2.1 Supported physical layers

EBI commands are typically sent/received over a UART interface. The default parameters for UART communications are 9600 baud/sec, 8 data bits, no parity, hardware flow control disabled.

The UART messages formatted using EBI can also be transported over USB connections (e.g., Embit evaluation boards feature a virtual COM emulation that allows to send EBI commands via USB) or over Ethernet (employing UART-to-Ethernet gateways like e.g., Embit EMB-GATE920T).

## 2.2 Packet format

All EBI binary packets have the following common structure:

| Field | Packet length | Message ID | Payload (format specific for each Message ID) | Checksum |
|---|---|---|---|---|
| Length | 2 Bytes | 1 Byte | Variable | 1 Byte |

The “Packet length” field specifies the number of bytes for the packet including all fields (i.e., including the “packet length”, “message ID”, “payload” and “checksum”).

The “Message ID” field specifies the type of message and how the payload field has to be processed. The most significant bit indicates, if set to 1, that the packet is a response to the request identified by the lower 7 bits or is a notification (i.e., a packet that does not require any response).

The “Payload” field is formatted differently for each message ID and contains details on the actions to be performed. As mentioned above, please refer to EBI WMBus-specific Documentation”, “EBI 802.15.4-specific Documentation” or EBI ZigBee-specific Documentation” for such information.

The “Checksum” field is the 8 bit sum of every byte in the packet (except for the Checksum field itself!) with initial value 0x00.

An example of EBI packet is the following (using hexadecimal notation):

```
0x00 0x06 0x32 0x00 0x04 0x3C
```

This packet has a length of 6 bytes and represents a request with message ID = 0x32 (corresponding to the "network scan" command, see e.g., "EBI 802.15.4-specific documentation"), with a payload of two bytes equal to 0x00, 0x04. Note that the checksum byte 0x3C is equal to 0x06 + 0x32 + 0x04.

The data structure holding the EBI packet example above, in C language, could be the following:

```
struct EBIPacketNetworkScan
{
      uint8_t    length_msb;
      uint8_t    length_lsb;
      uint8_t    messageId;
      uint8_t    payload[2];
      uint8_t    checksum;
};
```

A simple example function that checks if a "EBIPacketNetworkScan" has been received correctly could be:

```
bool check_packet(EBIPacketNetworkScan* pkt)
{
    uint8_t computed_checksum = 0;
        /* note that an uint8_t is used to ensure
           that the sum is an 8-bit sum */
    computed_checksum += pkt->length_msb;
    computed_checksum += pkt->length_lsb;
    computed_checksum += pkt->messageId;
    computed_checksum += pkt->payload[0];
    computed_checksum += pkt->payload[1];

    return computed_checksum == pkt->checksum;
}
```

Note that all packet fields longer than one byte are transmitted in big endian order (i.e., the most significant byte is sent first over the UART).

Moreover, to each EBI command sent in the direction "host → module" corresponds an EBI response in the opposite direction[1]; the host is required to wait for the EBI response before sending the next EBI command: the wireless module implementing EBI can only process one command at a time. Every response is identified by the most significant bit set to one; for example, the response to the message ID 0x32 has the code 0xB2.

1 The Embit wireless module will NOT send an EBI response only in two cases: a) the EBI command sent is not well-formatted (or the checksum is wrong), b) the EBI command is unsupported by that specific module.

To keep things simple and avoid buffer underrun problems **the user must avoid sending packets until the response for the previous packet is received**.

## 2.3 Packet types

EBI commands can be classified in two different types of packets:

1. **Command execute packets**: these packets usually do not have a payload; used for command execution have a response in which the first byte is typically an "execution status byte" (see Section 2.4) indicating if the command has been executed successfully;
2. **Read/write parameter packets**: these packets typically have a payload formatted in two different ways depending if the parameter is to be read or write.
   To *read* a parameter the packet is sent with an empty payload and the module will respond with the current value of the parameter in the response payload. Unsupported parameters will have a response with an empty payload.
   To *write* a parameter the command is sent with the parameter value in the payload and the module will respond with a single byte in the "execution status byte" format. When writing unsupported parameters, the module will always return "unsupported" in the response payload.

## 2.4 Execution status byte

Several responses to EBI commands (in particular responses to "Command execute packets") have a payload with a field called "execution status byte". This status byte must be interpreted as an acknowledge return value and has the following generic meanings:

0x00 = Success
0x01 = Generic error
0x02 = Parameters not accepted
0x03 = Operation timeout
0x04 = No memory
0x05 = Unsupported
0x06 = Busy

## 2.5 Available commands

A list of all the binary commands of the EBI protocol is provided here, together with the indication of which EBI variant (EBI-WMBus, EBI-802.15.4, EBI-ZigBee) implements it:

| Message ID | Command Description | Supported by: | | |
|---|---|---|---|---|
| | | EBI 802.15.4 | EBI ZigBee | EBI W-MBus |
| 0x01 | Device Information | V | V | V |
| 0x04 | Device state | V | V | V |
| 0x05 | Reset | V | V | V |
| 0x06 | Firmware version | V | V | V |
| 0x07 | Restore to factory default settings | V | V | V |
| 0x08 | Save settings | V | V | V |
| 0x09 | Serial port configuration | V | V | V |
| 0x10 | Output power | V | V | V |
| 0x11 | Operating channel | V | V | V |
| 0x12 | Active channel mask | V | V | |
| 0x13 | Energy save | V | V | V |
| 0x14 | Force sleep | V | V | |
| 0x15 | Force data poll | V | V | |
| 0x1D | Timer control | | | |
| 0x1E | ADC control | | | |
| 0x1F | GPIO control | | | |
| 0x20 | Physical address | V | V | V |
| 0x21 | Network address | V | V | V |
| 0x22 | Network identifier | V | V | |
| 0x23 | Network role | V | V | V |
| 0x24 | Network automated settings | V | V | V |
| 0x25 | Network preferences | V | V | |
| 0x26 | Network security | | V | V |
| 0x30 | Network stop | V | V | V |
| 0x31 | Network start | V | V | V |
| 0x32 | Network scan | V | | |
| 0x38 | Add endpoint | | V | |
| 0x39 | Remove endpoint | | V | |
| 0x40 | Associated addresses | V | V | |
| 0x41 | Associating device | V | V | |
| 0x50 | Send data | V | V | V |
| 0x60 | Received data | V | V | V |
| 0x70 | Enter bootloader | V | V | V |
| 0x71 | Set bootloader options (*) | V | V | V |
| 0x78 | Erase memory (*) | V | V | V |
| 0x7A | Write memory chunk (*) | V | V | V |
| 0x7B | Read memory chunk (*) | V | V | V |
| 0x7F | Commit firmware (*) | V | V | V |

(*) Available only during *bootloading* phase.

# 3 Annex

## 3.1 Disclaimer

The information provided in this and other documents associated to the product might contain technical inaccuracies as well as typing errors. Regulations might also vary in time. Updates to these documents are performed periodically and the information provided in these manuals might change without notice. The user is required to ensure that the documentation is updated and the information contained is valid. Embit reserves the right to change any of the technical/functional specifications as well as to discontinue manufacture or support of any of its products without any written announcement.

## 3.2 Trademarks

Embit is a registered trademark owned by Embit s.r.l.

All other trademarks, registered trademarks and product names are the sole property of their respective owners.