FlexiPanel

MACdongle™

IEEE 802.15.4 MAC layer firmware for UZBee USB adapter

Summary

MACdongle allows USB-enabled devices to implement the IEEE 802.15.4 communications protocol for low data-rate wireless personal area networks. It is ideal for OEMs who need to add USB capability to their ZigBee networks. It incorporates an FCC / CE certified IEEE 802.15.4 transceiver and USB interface.

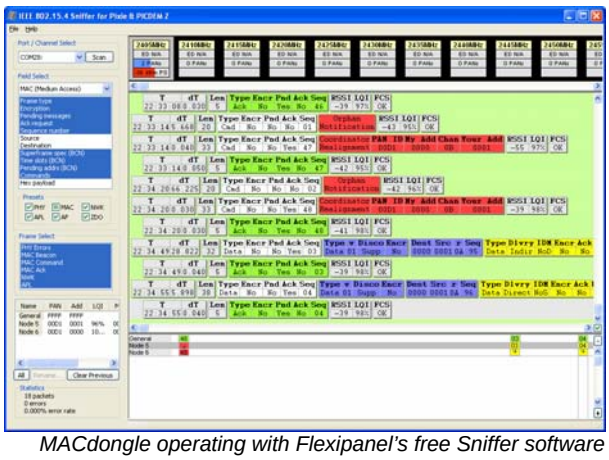
MACdongle provides a gateway between IEEE 802.15.4 systems and Microsoft Windows PC software. As a derivative of the FlexiPanel USBoot family, its firmware may be reprogrammed interchangeably with any other compatible firmware via the USB port.

Firmware Features:

- *Fully compatible with free-of-charge Sniffer software for Windows PCs*
- *MAC-layer services include:*
  - *Data packet transmission & reception*
  - *Associate & disassociate*
  - *Beacon notification*
  - *Orphan notification*
  - *Energy density scan*
  - *Active scan*
  - *Passive scan*
  - *Orphan scan*
  - *COMM status*
  - *PIB set & get*
  - *Sync request / sync loss indication*
  - *Poll request*
  - *Transmission purge*
- *PHY-layer services include:*
  - *Data packet hook for packet sniffing*
- *Device-layer services include:*
  - *LED control*
  - *Pushbutton indication*
  - *Error report*
  - *Device information report*
  - *One-time MAC address configuration*

Hardware Features:

- *USBoot bootloader allows firmware interchange between:*
  - ***MACdongle™*** *IEEE 802.15.4 MAC layer and sniffer*
  - ***Zongle™*** *ZigBee RFD APS layer*
  - ***StarLite USB™*** *transparent IEEE 802.15.4 communications*
- *2.4GHz IEEE 802.15.4 RF module*
- *FCC / CE / IC compliant*
- *Signature 'G' antenna, free-space range 120m, compact, low 'hand-effect' design*
- *Bind button, status LED*
- *56mm x 20mm x 9mm*



MACdongle operating with Flexipanel's free Sniffer software

Ordering Information

Table 1. Ordering information	
Part No	***Description***
UZBee	USB IEEE 802.15.4 radio adapter

Manufactured to ISO9001:2000

Contents

Summary ***1***

Contents ***2***

MACdongle Overview ***3***

Device Types 3
Frequency channels 3
MAC-address communications 3
Short-address communications 3
Sleep Management 4
Notation, Byte & Bit order 4
Release notes, version 0B40011110352015110 6tt 4
Bibliography 4

Installation Procedures ***5***

Service Pack Installation 5
MACdongle Installation 5
.inf File Customization 6

Usage examples ***7***

Setting the MAC Address 7
Presence Detection 7
Disabling Timeouts 7
Sending Packets (long address mode) 8
Receiving Packets (long address mode) 8
Scanning 8
Starting a network 9
Permitting joining 9
Joining a network 10
Permitting rejoining 10
Rejoining a network 11
Sending Packets 11
Receiving Packets In A Network 12
Sniffing 12

Message Reference ***13***

Format 13
Device Messages 13
Error indication (DERI) *13*
Enquire MAC Address request (DMCR) *13*
Enquire MAC Address confirm (DMCC) *14*
Set MAC Address request (DSMR) *14*
Set MAC Address confirm (DSMC) *14*
Version request (DVRR) *14*
Version confirm (DVRC) *14*
LED Display request (DLDR) *15*
LED Display confirm (DLDC) *15*
Pushbutton indication (DPBI) *15*
Suppress Timeout request (DHTR) *15*
Suppress Timeout request (DHTC) *15*
Continue indication (DCTI) *16*
PHY Messages 16
Physical-layer data indication (PDAI) *16*
MAC Messages 17
MCPS-DATA.request (MDAR) *17*
MCPS-DATA.confirm (MDAC) *17*
MCPS-DATA.indication (MDAI) *18*
MCPS-PURGE.request (MPGR) *18*
MCPS-PURGE.confirm (MPGC) *18*
MLME-ASSOCIATE.request (MASR) *19*
MLME-ASSOCIATE.indication (MASI) *19*
MLME-ASSOCIATE.response (MASS) *20*
MLME-ASSOCIATE.confirm (MASC) *20*
MLME-DISASSOCIATE.request (MDSR) *20*
MLME-DISASSOCIATE.indication (MDSI) *20*
MLME-DISASSOCIATE.confirm (MDSC) *21*
MLME-BEACON-NOTIFY.indication (MBNI) *21*
MLME-GET.request (MGTR) *21*
MLME-GET.confirm (MGTC) *22*
MLME-GTS.request (MTSR) *22*
MLME-GTS.confirm (MTSC) *22*
MLME-GTS.indication (MTSI) *22*
MLME-ORPHAN.indication (MORI) *22*
MLME-ORPHAN.response (MORS) *23*
MLME-RESET.request (MRSR) *23*
MLME-RESET.confirm (MRSC) *23*
MLME-RX-ENABLE.request (MRXR) *23*
MLME-RX-ENABLE.confirm (MRXC) *23*
MLME-SCAN.request (MSCR) *23*
MLME-SCAN.confirm (MSCC) *24*
MLME-COMM-STATUS.indication (MCSI) *25*
MLME-SET.request (MSTR) *25*
MLME-SET.confirm (MSTC) *26*
MLME-START.request (MSRR) *27*
MLME-START.confirm (MSRC) *27*
MLME-SYNC.request (MSYR) *27*
MLME-SYNC-LOSS.indication (MSLI) *27*
MLME-POLL.request (MPLR) *27*
MLME-POLL.confirm (MPLC) *28*

Reference ***29***

Radio Frequency *29*
Electrical *29*
Mechanical *29*
Regulatory *29*

Contact Information ***29***

MACdongle Overview

MACdongle is a standard non-beacon implementation of the IEEE 802.15.4 communications specification. This overview provides a general introduction to non-beacon IEEE 802.15.4 networks, but only in sufficient detail to implement working networks with MACdongle devices and other devices in the FlexiPanel IEEE 802.15.4 firmware range. The Usage Examples section that follows shows examples of actual commands being sent to MACdongle.

For a broad introduction to the different types of RF firmware available from FlexiPanel, refer to *DS500, RF Transceiver Selection Guide*.

Device Types

Three types of device are implemented, each with a different firmware build. It is not possible to switch between device types at runtime.

- ***UMDC:*** MACdongle Coordinator, creates a IEEE 15.802.4 network
- ***UMDF:*** MACdongle Full Function Device, Rx-On-When-Idle, participates in an IEEE 802.15.4 network, cannot sleep
- ***UMDS:*** MACdongle Full Function Device, Rx-Off-When-Idle, participates in an IEEE 802.15. 4, network can sleep

Note that for UZBee revision number UZBr7 (indicated in the antenna area of the PCB, the versions with the suffix ***–UZBr7*** should be used.

Frequency channels

UZBee operates in the 2.4GHz frequency band on sixteen channels numbered 11 to 26 (0x0B to 0x1A in hex).

MAC-address communications

In theory, devices can communicate at any time by addressing each other by their MAC address (8 bytes, otherwise known as the long address). However, this requires that frequency channel and the MAC address of every device be known in advance, and also than no devices sleep. This is not practical for commercial products, but can be useful for one-off custom designs, since no network needs to be started or joined.

Short-address communications

In order to be able to share airspace, and to permit devices to learn who to talk to at installation time rather than at the factory, commercial systems need use short-addressing. The coordinator will start and assign itself a short address (2 bytes) and a network-wide PAN ID (2 bytes). Then other devices ask to join the network and the coordinator will remember their long address and allocate a short address in return. The long address is only used thereafter if the network reinitializes and devices need discover the new frequency and PAN ID. Typically the joining process is only permitted to occur by pressing a ‘bind’ button on the coordinator. This ensures the correct device joins the correct network in a secure manner.

Sleep Management

The ***UMDS*** version of the firmware is allowed to "sleep". Practically speaking, it doesn't make much sense to implement sleep on the USB device. However, this does permit the PC to be not-always-present or not-always-on.

Sleepy devices can only receive unicast data from the coordinator; to do so they must specifically request it using the poll request command. The coordinator will also need to remember which devices sleep in order to know whether or not to cache their messages.

Notation, Byte & Bit order

All numbers in this documentation are in decimal unless prefixed with 0x, in which case they are hexadecimal. Index counting starts at zero, so the first byte of a message is byte zero.

Multi-byte data is transmitted least-significant byte first ('little-endian'), as is standard in the IEEE 801.15.4 specification.

Release notes, version 0B40011110352015110 6tt

tt refers to the device type. Refer to the **DVRC** message for details. In this release, security and beacon networks are not supported. The external oscillator is used, so the 16MHz crystal must be fitted.

Bibliography

IEEE 802.15.4 specification, downloadable from *www.ieee.org*.

ZigBee for Applications Developers, white paper downloadable from *www.flexipanel.com*.

ZigBee Specification, downloadable from *www.zigbee.org*.

CC2420 Data Sheet, downloadable from *www.chipcon.com*.

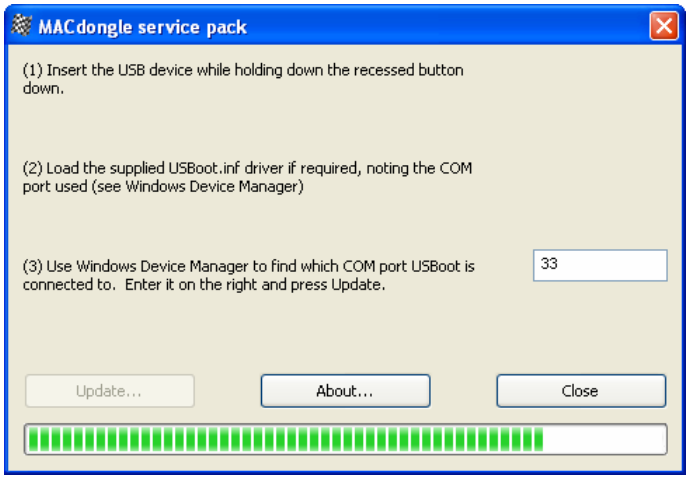
Installation Procedures

Service Pack Installation

MACdongle is firmware for the UZBee USB IEEE 802.15.4 transceiver. This section explains how to load the MACdongle firmware onto UZBee using FlexiPanel's *USBoot* bootloading procedure. You can skip the *Service Pack Installation* section if you know MACdongle is already loaded. To install MACdongle:

1. Insert the UZBee into a USB port while holding the recessed pushbutton down. The LED should flash. If this is the first time you have run any USBoot product, you will be asked for the `USBoot.inf` driver information file, which may be downloaded from *www.flexipanel.com.*

2. Download and unzip the MACdongle development kit from *www.flexipanel.com*. Run the appropriate application for the firmware you want (e.g. `UMDC.exe` – see MACdongle Description for further information).



3. To determine which COM port number has been assigned to USBoot:

    a. Click on *Start*
    b. Right click on *My Computer*, select *Properties*
    c. Click on the *Hardware* tab, press the *Device Manager* button.
    d. Open the *Ports (COM & LPT)* section. You will see the device name *USBoot* and the COM port number.

4. Enter the COM port number in the box provided.

5. Press the Update button. The firmware will load automatically and then start running as if you had just inserted MACdongle into a USB slot.

Note that when installed in this way, the MAC address is not preloaded and you may have to specify it using the ***+DSMR*** command. (Application software such as FlexiPanel's Sniffer should do this for you.) If you use the service pack when you load MACdongle into OEM products for distribution to your customers, it is your responsibility to set the MAC address on behalf your customers. Contact us to obtain an allocation of addresses if you wish to have a contiguous block.

MACdongle Installation

If this is the first time you have run MACdongle, you will be asked for the `MACdongle.inf` driver information file, which may be downloaded from *www.flexipanel.com.* The file will be required by users of your products, so it should be packaged with the product or be made readily available on a web-site.

When the USB device is first plugged into a new USB port, Windows will request the `.inf` file. A dialog box will note that the driver is not certified, to which you should select *Continue Anyway*. The USB device will be assigned an unused COM port number. To determine which number has been assigned:

1. Click on *Start*
2. Right click on *My Computer*, select *Properties*
3. Click on the *Hardware* tab, press the *Device Manager* button.
4. Open the *Ports (COM & LPT)* section. You will see the device name MACdongle and the COM port number.

Your users will also need these instructions.

If you plug the MACdongle into a different socket, the driver may need to reinstall itself a second time, but this time the computer should find `.inf` file by itself. A different COM port will be assigned, but otherwise the MACdongle will function as normal.

MACdongle should be plugged in before the application attempts to open the COM port. Failure to do so may require the application to be closed and MACdongle to be re-inserted.

.inf File Customization

MACdongle uses the standard CDC Windows drivers for data transfer. During installation, only a `.inf` driver information file is needed. This is called `MACdongle.inf`, and can be downloaded from *www.flexipanel.com*.

If you wish to develop your own-brand applications for MACdongle, the `.inf` driver information file may be customized. It is a text file that may be modified in Windows Notepad as follows. (These steps are optional – you may distribute the `MACdongle.inf` file as-is.)

- Modify MFGNAME and DESCRIPTION to suit your application.
- The filename may be modified but the `.inf` suffix should remain.

Usage examples

The following examples show how MACdongle can be used. To test these functions, use the HyperTerminal terminal emulator available in Microsoft Windows to communicate with MACdongle.

Application software interfacing with MACdongle should treat it as a COM port. Since data flow is asynchronous and unpredictable, data transmission should not block data reception and vice versa. For Microsoft Windows programming, for example, it is vital that overlapped file I/O be used.

Note that data are expressed little-endian, i.e. multi-byte values need to be interpreted in reverse-byte order. For example `940D` represents the word value 0x0D94.

Setting the MAC Address

If presence detection generates an error indicating the MAC address is not set, the MAC address should be set.

+DVRR	(request version or any command)
+DERI=04	(confirmation – error, MAC address not set)
+DSMR=0600004138C81500	(request to set MAC address)
+DSMC	(confirmation – confirms MAC address set)

Note that the MAC address is specified Little-Endian, i.e. the MAC address specified in the example would normally be written as ***0015C83841000006***. The MAC address is stored in non-volatile memory and only needs to be set once.

You may only use a MAC address which you have been assigned. If you do not have a MAC address allocation, contact FlexiPanel Ltd.

Presence Detection

The presence of MACdongle can be checked by issuing a version number request. The last digit is the firmware device type (1 = UMDC, 2 = UMDF, 3 = UMDS)

+DVRR	(request version)
+DVRC=0B400113103520151106O1	(confirmation – version number may vary)

Disabling Timeouts

If you're evaluating MACdongle by typing in commands, life will be a lot easier if you disable timeouts, so you're not in a rush to type instructions. Caution, however – you must still respond when required rather than issuing another instruction, otherwise the stack will get in a mess.

+DHTR	(request that timeouts be disabled)
+DHTC	(confirmation – disabled)

Sending Packets (long address mode)

The simplest way to send data, provided everybody knows each others' MAC addresses and all devices are operating on the same frequency, is to address each other using the MAC address. Packets of data sent using the MDAR request:

```
+MDAR=04000101FFFF0303989898989898983434343434343434FFFF8890abcdef
+MDAR=04000101FFFF0303989898989898983434343434C81500FFFF8890abcdef
+MDAR=04000101FFFF030398989898989898757575757 5C81500FFFF8890abcdef
```

(request send data 90ABCDEF, length 04 from 9898989898989898 to 3434343434343434, long addr mode (0303))

```
+MDAC=00000101FFFF0303989898989898983434343434343434FFFF5F
```

(confirmation – data sent)

Transmission is acknowledged. If no reply is received, the confirmation will report failure.

Note that the PAN ID is set to FFFF, indicating no particular PAN. Prior to transmission, ensure all devices select the same frequency channel. The data handle in +MDAR is ignored; the current *macDSN* value is used and then incremented.

Receiving Packets (long address mode)

Received packets are normally processed by the MAC layer and generate a MAC-level indication message, *e.g.*:

```
+MDAI=04001601FFFF0303434343434343434989898989898989FFFF19E890ABCDEF
```

(indication – 90ABCDEF received from 3434343434343434, long addr mode (0303))

Note that a sleepy end device would have to poll for messages; refer to the *Receiving Packets In A Network* section for more details; the poll would have to specify the coordinator long address in such a case.

Scanning

An active scan for devices on different channels can be used to determine which networks are operating on which frequencies. Example:

```
+MSCR=00000000000010800F8FF07
```

(request active scan, all channels 0x0B – 0x1A)

```
+MSCC=EA00000000000108FF0700F8
```

(confirmation – scan result, nothing found)

Another example:

```
+MSCR=00000000000010800F8FF07
```

(request scan)

```
+MBNI=03000C01940D020B0000FF07B98084B043D3120110435A00FF4F44E0001100
```

(indication – response from network 940D)

+MSCC=00011201120101 0BFF0700F800 0B 940D 0000 FF07B98084B0FF4F43D31201E00000
(confirmation – 01 network found, channel 0B,
PAN ID 940D, coordinator short address 0000)

Starting a network

To start a network, the coordinator must select an operating frequency and PAN ID based on the results of an active scan. The frequency chosen should be a little-used one and certainly should not have a network with the same PAN ID. The network is then started by setting the device's short address and then issuing a +MSRR start request:

+MSTR=530000	(Set short address to 0000)
+MSTC=0053	(Confirm set short address)
+MSRR=0100000034120B00000000000F0F	(Start PAN ID 1234 on channel 0B)
+MSRC=00	(Confirm start PAN)

Once you have started the network, you should be able to locate it using a scan from another device such as a sniffer.

Permitting joining

To allow networks to join a coordinator's network, its *AssocPermit* flag must be set:

+MSTR=4101 (set AssocPermit to 01)

When a device requests to join, (see next section, *Joining a network*,) an association indication will be generated:

+MASI=8E001401FFFF0302343434343434343400 0040BDE4306B2C3412E9 (Associate indication, including *CapabilityInformation*,
and *MAC address of associating device*)

The coordinator must then decide whether or not to let the device join and issue a MASS response. If it is allowed to join, it is assigned a short address and the status value is zero. If not, the status value should be 0x01 if the PAN is "at capacity" or 0x02 if association is denied.

+MASS=00 0000000000 7856 0000000000000000 3434343434343434
+MASS=00 0000000000 7856 0000000000000000 7575757575C81500
(Associate indication, including *Status*, allocated
short address 0x5678 and *assoc device MAC address*)

The short address and long address of the joining device must then be stored in a network membership table so that the device may rejoin as an orphan at a later date. In addition, the the *CapabilityInformation* byte will have to be stored if there are sleepy devices on the network. Otherwise the coordinator will no know whether it can transmit data immediately to the device ("direct", i.e. RxOnWhenIdle) or whether the data should be cached until the device asks for it ("indirect", i.e. RxOffWhenIdle).

Joining a network

To join a network, a device must first discover it using an active scan. Only networks with their *AssocPermit* bit set can be joined.

Once a network has been selected, an association request is made:

+MASR=8E00000000000B020000000000000000000000000000003412

(Associate request, including *CapabilityInformation*, *Channel*, *Coordinator short address*, *PAN ID*)

The recommended values for CapabilityInformation are 0x8E for Rx-on-when-idle devices and 0x82 for Rx-off-when-idle devices. (These values may change when security is added.)

Note that if you have suppressed timeouts (+DHTR) you will need to send a poll request to receive any response made by the coordinator, e.g.

+MPLR=00000000000000020000000000000000000000000000003412

(Poll request, including *Coord short address*, *PAN ID*)

A response should then be received from the coordinator.

+MASC=00001A0134120000 (Association confirmed – Successful (00))

If joining is successful, the short address, coordinator short address and PAN ID will automatically be set.

Permitting rejoining

If a device has already joined a network but is reinitializing and looking for its parent, (see next section, *Rejoining a network*), coordinators in range will receive an orphan indication:

+MORI=00001301FFFF0302343434343434343434FFFF343434343434FFFF5D

(Orphan indication, including *MAC address of orphan device*)

The +MORI orphan indication should be responded to with an +MORS orphan response. If the orphan device is a member of the coordinator's network, the coordinator should reply by reminding it of its short address:

+MORS=0100000000000000343434343434343434347856
+MORS=0100000000000000757575757575C815007856

(Orphan response, including *Yes you're mine*, and *MAC address and short address of orphan device*)

Even if the orphan device is a not member of the coordinator's network, the coordinator should respond to the stack:

+MORS=0000000000000000343434343434343434FFFF

(Orphan response, including *No it's not mine*, and *MAC address of orphan device and null short address*)

Rejoining a network

If a device has already joined a network but is reinitializing, it does not need to repeat the join procedure. It can simply perform an orphan scan to see if its coordinator acknowledges it.

+MSCR=000000000000030800080000 (request orphan scan, channel 0x0B only)

+MSCC=00000000000000303FF070000 (confirmation – scan result, success)

+MSCC=EA00000000000308FFF7FFFF (confirmation – scan result, failure)

Note that if you specified +DHTR, each channel will be scanned for 20 seconds to give you time to paste a response into HyperTerminal on the coordinator. This is why only channel 0x0B is scanned in this example– it would take ages otherwise!

If rejoining is successful, the *short address*, *coordinator short address* and *PAN ID* attributes will automatically be set.

Sending Packets

Once a network is established, packets are sent using the MDAR request. If the recipient does not sleep (RxOnWhenIdle), the packet should be sent directly:

+MDAR=040001013412020200000000000000000100000000000034128890abcdef

+MDAR=040001013412020200000000000078560000000000003412 8890abcdef

+MDAR=040001013412020278560000000000000000000000000034128890abcdef

(request send data 90ABCDEF from Addr 0000 / PAN 1234 to Addr 0001 / PAN 1234, direct, Aclnowledged (01))

+MDAC=0000010034120202000000000000000010000000000000341253

(confirmation – data sent)

If the recipient does sleep (RxOffWhenIdle), a coordinator may send the packet indirectly, i.e. cached until the device wakes to poll for messages:

+MDAR=040001053412020200000000000000000100000000000034128890abcdef

+MDAR=040001053412020278560000000000000000000000000034128890abcdef

(request send data 90ABCDEF from Addr 0000 / PAN 1234 to Addr 0001 / PAN 1234, indirect, Aclnowledged (05))

+DCTI (indication request complete)

+DCTI indicates only that the packet is sequenced. Actual confirmation will arrive later when the packet is sent or times out:

+MDAC=0000010034120202000000000000000010000000000000341253

(confirmation – data sent)

In the meantime, further requests may be made; the number of messages that may be cached like this varies with message length; the absolute maximum is 32.

To broadcast, use the short address FFFF:

+MDAR=04000100341202020000000000000000FFFF0000000000034128890abcdef

(request broadcast 90ABCDEF from Addr 0000 / PAN 1234 to PAN 1234, direct, unacknowledged (00))

Broadcast packets should be neither indirect nor acknowledged.

Receiving Packets In A Network

Received packets are normally processed by the MAC layer and generate a MAC-level indication message, *e.g.*:

+MDAI=04500A013412020200009898989898980100343434343434341225E890ABCDEF
(indication –packet received)

If the device is a sleepy device, it must first request data from the coordinator using a poll request. Typically, it will issue poll requests on a regular basis while awake:

+MPLR=0000000000000002000000000000000000000000000003412
(request – poll PAN coordinator indirectly)

+MDAI=04500A013412020200009898989898980100343434343434341225E890ABCDEF
(indication – packet received)

+MPLR=0000000000000002000000000000000000000000000003412
(request – poll PAN coordinator indirectly)

+MPLC=EB (confirmation – no more data)

Sniffing

Address filtering can be turned off by enabling promiscuous mode. In this mode, received packets are not processed by the MAC layer but instead generate PDAI messages:

+MSTR=000B (request channel 0B, 2405MHz)
+MSTC=0000 (confirmation – channel set)
+MSTR=5101 (request promiscuous mode)
+MSTC=0051 (confirmation – promiscuous mode set)
+PDAI=0512005FF3EC (indication – packet received)

This mode is used by the Pixie Sniffer application to provide a sniffing tool. Any version of MACdongle can be used as a sniffer detector.

Message Reference

Format

Messages sent to and received from MACdongle take the form of ASCII message strings with the following general format:

***+XXXX=hhhhhhhh*<CR><LF>**

All messages begin with the **+** character followed by the four-letter command or response code ***XXXX***, followed by the **=** character. If any additional data accompanies the message, it usually follows as a series of bytes each represented two hexadecimal digits ***hh***. Multi-byte integers are parsed *little-Endian*, i.e. least significant byte first. If no additional data accompanies the message, the **=** character may be omitted. Finally, the string is terminated with a carriage return character **<CR>** and optionally a linefeed **<LF>** character. Extra **<CR>** and/or **<LF>** characters are permitted between messages. Inline editing (*e.g.* pressing backspace) is not supported.

Do not send a message until processing of the previous message is complete. After processing a command, MACdongle ignores all characters until a **+** character is received. To abandon after starting it by entering **+**, press ***Z*** until a ***DERI*** syntax error is generated. After processing a command, MACdongle ignores all characters until a **+** character is received.

Device Messages

Messages starting with a ***D*** character relate to device settings and values.

Error indication (DERI)

DERI indicates a device-level error occurred. Example:

+DERI=01

The following error values may be reported:

Value	Interpretation
01	Syntax error – message not recognized
02	Syntax error – parameters syntax invalid
03	Syntax error – parameters count invalid
04	MAC address not valid – send **+DRMC=** message first
05	MAC address has already been set, cannot be changed
06	Message not currently supported

Enquire MAC Address request (DMCR)

DMCR enquires the MAC address of MACdongle. MACdongle will generate a DMCC confirmation containing the MAC address. Example:

+DMCR

Enquire MAC Address confirm (DMCC)

DMCC confirms the 8-byte MAC address of the MACdongle. Example:

***+DMCC*=0600004138C81500**

Note that the MAC address is specified Little-Endian, i.e. the MAC address specified in the example would normally be written as ***0015C83841000006***.

Set MAC Address request (DSMR)

DSMR specifies a MAC address for MACdongle. This command may only be completed once, and since it will normally be set for you, this command will usually generate an error. If you use a bootloader to program the MACdongle firmware, you should first identify and note down the current the MAC address and then re-set it using this command. You may only use the MAC address which the hardware has been allocated. For R&D purposes, MAC addresses in the range `0015C83841000000` to `0015C8384100FFFF` may be specified.

MACdongle will generate a DSMC confirmation in response. Example:

***+DSMR*=0600004138C81500**

Note that the MAC address is specified Little-Endian, i.e. the MAC address specified in the example would normally be written as ***0015C83841000006***.

Set MAC Address confirm (DSMC)

DSMC confirms that a DSMR request completed without error. It has no arguments. Example:

+DSMC

Version request (DVRR)

DVRR requests firmware version information from MACdongle. MACdongle will generate a DVRC confirmation in response. Example:

+DVRR

Version confirm (DVRC)

The DVRC confirmation is generated by MACdongle in response to a DVRR request. Example:

+DVRC=0B400111103520240406 01

The ID takes the form ***+DVRC=UUUUPPPPVVVVVVDDMMYYtt***, where the digits are as follows:

UUUU	USB Vendor ID (0B40 always)
PPPP	USB Product ID (0111 indicates MACdongle firmware is loaded)

VVVVVV MACdongle version number
DDMMYY Firmware release date
tt Device type (01 = ***UMDC***, 02 = ***UMDF***, 03 = ***UMDSC***)

LED Display request (DLDR)

DLDR sets the state of the MACdongle LED. MACdongle will generate a DLDC confirmation in response. Examples:

+DLDR=00 (LED off – initial state)

+DLDR=01 (LED on)

LED Display confirm (DLDC)

DLDC confirms that a DLDR request completed without error. It has no arguments. Example:

+DLDC

Pushbutton indication (DPBI)

DPBI indicates a change of state of the pushbutton. It has a one byte argument, which is nonzero if the button was pressed and zero if the button was released. Examples:

+DPBI=01 (Pushbutton was pressed)

+DPBI=00 (Pushbutton was released)

Suppress Timeout request (DHTR)

Certain commands (association indications and orphan indications) timeout quickly if no response is received from the host. If you're evaluating PixieMAC using HyperTerminal, you haven't a hope of typing in responses in time. The +DHTR command extends timeouts until the time the module is powered up, allowing easier evaluation with HyperTerminal. The extension is infinite in the case of association indications and 20 seconds in the case of orphan indications. Caution, however – you must still respond when required rather than issuing another instruction, otherwise the stack will get in a mess. (Also note that you may have to send out poll requests to actually receive a response which would normally have timed out – see the usage examples.)

Example:

+DHTR

Suppress Timeout request (DHTC)

DHTC confirms that a DHTR request completed without error. It has no arguments.

Continue indication (DCTI)

When issued in response to an associate response (+MASS) or orphan response (+MORS), DCTI indicates that the message has been processed. The next command may now be sent.

When issued in response to an indirect data request (+MDAR), DCTI indicates that the message has been cached waiting for the recipient to wake up. The next command may now be sent; +MDAC confirmation of this request will follow when the message is transmitted or times out; it may be identified by its *msduHandle*.

PHY Messages

Messages starting with a ***P*** character relate to the MAC layer.

Physical-layer data indication (PDAI)

When promiscuous mode is set, (refer to MSTR command), no address filtering is applied and all received packets are intercepted at the PHY level and converted direct to PDAI messages. The FCS (checksum) field will have been already verified and, contrary to the IEEE 802.15.4 specification, the first and second bytes of the FCS field will contain CC2420-defined RSSI and LQI values.

Command format		***+PDAI=****{1+Length bytes}*
Length	1 byte	Packet length in bytes (PHR)
Data	*Length*	PHY payload

MAC Messages

Messages starting with an ***M*** character relate to the MAC layer.

MCPS-DATA.request (MDAR)

MDAR requests a data packet be transmitted.

Command format		***+MDAR=****{ msduLength + 27 bytes}*
msduLength	1 byte	Length of payload *msdu* to follow
Filler1	1 byte	*(fill with 00)*
FrameType	1 byte	0x00 = Beacon frame 0x01 = Data frame 0x02 = Ack frame 0x03 = MAC command frame
TxOptions	1 byte	Transmit options
SrcPanId	2 bytes	Source PAN ID
SrcAddrMode	1 byte	Source addressing mode
DstAddrMode	1 byte	Destination addressing mode
SrcAddr	8 bytes	Source address. (If *SrcAddrMode* specifies short addresses, ignore last 6 bytes.)
DstAddr	8 bytes	Destination address. (If *DstAddrMode* specifies short addresses, ignore last 6 bytes.)
DstPanId	2 bytes	Destination PAN ID
msduHandle	1 byte	Data handle
msdu	*msduLength* bytes	Payload to be transmitted
		IEEE 802.15.4 section 7.1.1.1

MCPS-DATA.confirm (MDAC)

MDAC responds to an MCPS-DATA.request. Note that +MDAC messages can be generated in response to internally generated MCPS-DATA.requests. If not received in response to an +MDAR command with a matching data handle, the confirmation should be ignored.

Command format		***+MDAC=****{27 bytes}*
status	1 byte	Result as enumeration
Filler1	25 bytes	*(ignore)*
msduHandle	1 byte	Data handle (equals macDSN number at time of ***+MDAR*** request)
		IEEE 802.15.4 section 7.1.1.2

MCPS-DATA.indication (MDAI)

MDAI indicates a data packet has been received. Note that CC2420 Auto-ACK is set, so packets are automatically acknowledged at the MAC level when not in promiscuous mode.

Command format		***+MDAI=****{ msduLength + 26 bytes}*
msduLength	1 byte	Length of payload *msdu*
SecurityUse	1 byte	Security indicator
Filler1	2 bytes	(ignore)
SrcPanId	2 bytes	Source PAN ID
SrcAddrMode	1 byte	Source addressing mode
DstAddrMode	1 byte	Destination addressing mode
SrcAddr	8 bytes	Source address. (If *SrcAddrMode* specifies short addresses, ignore last 6 bytes.)
DstAddr	8 bytes	Destination address. (If *DstAddrMode* specifies short addresses, ignore last 6 bytes.)
DstPanId	2 bytes	Destination PAN ID
mpduLinkQuality	1 byte	Link quality
ACLEntry	1 byte	*macSecurityMode* parameter
msdu	*msduLength* bytes	Payload received
		IEEE 802.15.4 section 7.1.1.3

MCPS-PURGE.request (MPGR)

MPGR requests a transmit packet gets purged from the queue.

Command format		***+MPGR=****{27 bytes}*
Filler1	26 bytes	*(fill with 00)*
msduHandle	1 byte	Data handle
		IEEE 802.15.4 section 7.1.1.4

MCPS-PURGE.confirm (MPGC)

MPGC reports on a purge operation.

Command format		***+MPGC=****{27 bytes}*
status	1 byte	Result as enumeration
Filler1	25 bytes	*(ignore)*
msduHandle	1 byte	Data handle
		IEEE 802.15.4 section 7.1.1.5

MLME-ASSOCIATE.request (MASR)

MASR requests an association with a PAN coordinator.

Command format		***+MASR=****{26 bytes}*
CapabilityInfo	1 byte	Capabilities of associating device Bit 0: True if Alt PAN Coordinator capable Bit 1: True if Full Function Device Bit 2: True if mains powered Bit 3: True if Rx-on-when idle (i.e. not sleepy) Bit 4: Reserved Bit 5: Reserved Bit 6: True if security capable Bit 7: True if short address is to be allocated
SecurityEnable	1 byte	True if security enabled
Filler1	4 byte	*(fill with 00)*
LogicalChannel	1 byte	Channel on which to associate
CoordAddrMode	1 byte	Coordinator addressing mode (0x02 = short, 0x03 = long)
Filler2	8 byte	*(fill with 00)*
CoordAddress	8 bytes	Coordinator address. (If *CoordAddrMode* specifies short addresses, ignore last 6 bytes.)
CoordPANid	2 bytes	Coordinator PAN ID
		IEEE 802.15.4 section 7.1.3.1

MLME-ASSOCIATE.indication (MASI)

MASI indicates the reception of an association request from another device. A MASS response ***must*** be generated if this indication is received.

Command format		***+MASI=****{28 bytes}*
CapabilityInfo	1 byte	Capabilities of associating device (see ***+MASR***)
SecurityEnable	1 byte	True if security enabled
Filler1	6 bytes	*(ignore)*
DeviceAddr	8 bytes	Associating device address
Filler2	11 bytes	*(ignore)*
ACLEntry	1 byte	*macSecurityMode* parameter
		IEEE 802.15.4 section 7.1.3.2

MLME-ASSOCIATE.response (MASS)

MASS initiates a response to a request for association with a PAN coordinator.

Command format		***+MASS=****{24 bytes}*
Status	1 byte	Result as enumeration
SecurityEnable	1 byte	True if security enabled
Filler1	4 bytes	*(ignore)*
AssocShortAddr	2 bytes	Short address allocated
Filler2	8 bytes	*(ignore)*
DeviceAddress	8 bytes	Address of device requesting association
		IEEE 802.15.4 section 7.1.3.3

MLME-ASSOCIATE.confirm (MASC)

MASC confirms the completion of an association request.

Command format		***+MASC=****{16 bytes}*
Status	1 byte	Result as enumeration
Filler1	7 bytes	*(ignore)*
DeviceAddr	8 bytes	Associating device address
		IEEE 802.15.4 section 7.1.3.4

MLME-DISASSOCIATE.request (MDSR)

MDSR requests disassociation from a PAN coordinator.

Command format		***+MDSR=****{16 bytes}*
DisassocReason	1 byte	Dissociation reason
SecurityEnable	1 byte	True if security enabled
Filler1	6 bytes	*(fill with 00)*
DeviceAddress	8 bytes	Device address
		IEEE 802.15.4 section 7.1.4.1

MLME-DISASSOCIATE.indication (MDSI)

MDSI indicates the reception of a disassociation request.

Command format		***+MDSI=****{28 bytes}*
DisassocReason	1 byte	Dissociation reason
SecurityEnable	1 byte	True if security enabled
Filler1	6 bytes	*(ignore)*
DeviceAddress	8 bytes	Device address
Filler2	11 bytes	*(ignore)*
ACLEntry	1 byte	*macSecurityMode* parameter
		IEEE 802.15.4 section 7.1.4.2

MLME-DISASSOCIATE.confirm (MDSC)

MDSC confirms the completion of a disassociation request.

Command format		***+MDSC=****{1 byte}*
status	1 byte	Result as enumeration
		IEEE 802.15.4 section 7.1.4.3

MLME-BEACON-NOTIFY.indication (MBNI)

MBNI indicates the reception of a disassociation request.

Command format		***+MBNI=****{sduLength + 23 bytes}*
sduLength	1 byte	Length of payload *sdu*
SecurityUse	1 byte	Security indicator
Filler1	2 bytes	*(ignore)*
CoordPanId	2 bytes	Coordinator PAN ID
CoordAddrMode	1 byte	Coordinator addressing mode
LogicalChannel	1 byte	Logical frequency channel 0x0B – 0x1A
CoordAddr	8 bytes	Coordinator address. (If *CoordAddrMode* specifies short addresses, ignore last 6 bytes.)
Timestamp	3 bytes	Timestamp
BSN	1 byte	Sequence number
Filler2	1 byte	*(ignore)*
SecurityFailure	1 byte	Security failure
GTSpermit	1 byte	Coordinator accepts GTS requests
SuperframeSpec	2 bytes	Superframe specification
LinkQuality	1 byte	Link quality
ACLEntry	1 byte	*macSecurityMode* parameter
AddrList	0 bytes	*(Beacon networks not currently supported)*
sdu	*sduLength* bytes	Beacon payload
		IEEE 802.15.4 section 7.1.5.1

MLME-GET.request (MGTR)

MGTR requests MAC and PHY attribute data. Refer to MLME-SET for a list of available attributes.

Command format		***+MGTR=****{1 byte}*
Attribute	1 byte	Attribute requested
		IEEE 802.15.4 section 7.1.6.1

MLME-GET.confirm (MGTC)

MGTC confirms attribute data.

Command format		***+MGTC=****{varies according to attribute}*
status	1 byte	Result as enumeration
Attribute	1 byte	Attribute (see table in **MSTR** section)
AttributeValue	*(see* **MSTC***)*	Attribute value
		IEEE 802.15.4 section 7.1.6.2

MLME-GTS.request (MTSR)

MRGT requests a guaranteed time slot allocation or deallocation. The command relates to beacon networks and is currently not supported.

MLME-GTS.confirm (MTSC)

MCGT confirms a request for a guaranteed time slot allocation or deallocation. The command relates to beacon networks and is currently not supported

MLME-GTS.indication (MTSI)

MGTC indicates that a guaranteed time slot allocation or deallocation has occurred. The command relates to beacon networks and is currently not supported

MLME-ORPHAN.indication (MORI)

MORI indicates the presence of an orphaned device. A MORS response ***must*** be generated indicating whether or not this device is the PAN coordinator for the orphan device.

Command format		***+MORI=****{28 bytes}*
Filler1	1 byte	*(ignore)*
SecurityUse	1 byte	True if security enabled
Filler2	6 bytes	*(ignore)*
OrphanAddr	8 bytes	Orphan device address
Filler3	11 bytes	*(ignore)*
ACLEntry	1 byte	*macSecurityMode* parameter
		IEEE 802.15.4 section 7.1.8.1

MLME-ORPHAN.response (MORS)

MORS responds to the presence of an orphaned device.

Command format		***+MORS=****{18 bytes}*
AssociateMember	1 byte	True if associated with this coordinator
SecurityEnable	1 byte	True if security enabled
Filler1	6 bytes	*(fill with 00)*
OrphanAddr	8 bytes	Orphan address
ShortAddr	2 bytes	Short address
		IEEE 802.15.4 section 7.1.8.2

MLME-RESET.request (MRSR)

MRSR requests that a reset operation is performed.

Command format		***+MRSR=****{1 byte}*
SetDefaultPIB	1 byte	If true, resets PIB attributes
		IEEE 802.15.4 section 7.1.9.1

MLME-RESET.confirm (MRSC)

MRSC confirms the result of a reset operation.

Command format		***+MRSC=****{1 byte}*
status	1 byte	Result as enumeration
		IEEE 802.15.4 section 7.1.9.2

MLME-RX-ENABLE.request (MRXR)

MRXR requests the receiver is enabled for a finite time. The command relates to beacon networks and is currently not supported.

MLME-RX-ENABLE.confirm (MRXC)

MRXC confirms a request for the receiver to be enabled for a finite time. The command relates to beacon networks and is currently not supported

MLME-SCAN.request (MSCR)

MSCR requests that a scan operation is performed. The following types of scan are possible:

Energy detect: Report radio activity density on channel, including Bluetooth and Wi-Fi, etc.

Passive scan: Listen for & report IEEE 802.15.4 activity on channel, including ZigBee, MailBox, etc.

Active scan: Issue beacon requests and listen for beacon responses from IEEE 802.15.4 devices on channel, including ZigBee, MailBox, etc.

Orphan scan: Issue orphan notification on channels and listen for claim of ownership (coordinator realignment) from a coordinator.

Command format		***+MRSC=****{12 bytes}*
Filler1	6 bytes	*(fill with 00)*
Scan type	1 byte	Scan type (00 = Energy detect, 01 = Active scan, 02 = Passive scan, 03 = Orphan scan
Scan duration	1 byte	Scan duration
Scan channels	4 bytes	Scan channels Bit 11 = true to scan channel 0x0B Bit 12 = true to scan channel 0x0C, etc
		IEEE 802.15.4 section 7.1.11.1

MLME-SCAN.confirm (MSCC)

MSCC confirms the result of a scan operation. Note that in the case of an orphan scan, a successful result will automatically set the *macCoordExtendedAddress*, *macCoordShortAddress*, *macPANId*, *macShortAddress* and *phyCurrentChannel* attributes to the correct values.

Command format		***+MSCC=****{size varies}*
status	1 byte	Result as enumeration
ResultListSize	1 byte	Number of results returned
Filler1	4 bytes	*(ignore)*
Scan type	1 byte	Scan type
Filler2	1 byte	*(ignore)*
Unscanned channels	4 bytes	Unscanned channels
EnergyDetectList†	*ResultListSize*	Energy detect list (1-byte values equal to [RSSI+128])
PanDescrList†	20 × *ResultListSize*	PAN Descriptor list (20-byte PAN Descriptor values)
† EnergyDetectList for energy detect scans only. PanDescrList for active and passive scans only.		
		IEEE 802.15.4 section 7.1.11.2

The PAN Descriptor List elements have the following format:

PAN Descriptor List		*{20 bytes}*
Flags	1 byte	Bit 0: 1 for *CoordAddrMode* long, 0 for short Bit 1: GTSPermit Bit 2: SecurityUse Bit 3: SecurityFailure Bits 4-7: ACLEntry
LogicalChannel	1 byte	Logical frequency channel 0x0B – 0x1A
CoordPANid	2 bytes	Coordinator PAN ID
CoordAddress	8 bytes	Coordinator address. (If *CoordAddrMode* specifies short addresses, ignore last 6 bytes.)
SuperframeSpec	2 bytes	Superframe specification
Timestamp	3 bytes	Timestamp
LinkQuality	1 byte	Link quality as reported by CC2420
Filler1	2 bytes	*(ignore)*

MLME-COMM-STATUS.indication (MCSI)

MCSI indicates a communications status. In many cases they are confirmations only and may be internally generated (e.g. in response to a MLME-POLL.request) and may be ignored.

Command format		***+MCSI=****{24 bytes}*
status	1 byte	Result as enumeration
Filler1	3 bytes	*(ignore)*
PanId	2 bytes	PAN ID (not populated since always macPIB.macPANId)
SrcAddrMode	1 byte	Source addressing mode (not populated since always 0x03=long)
DstAddrMode	1 byte	Destination addressing mode (not populated since always 0x03)
SrcAddr	8 bytes	Source address (not populated since always MAC address)
DstAddr	8 bytes	Destination address.
		IEEE 802.15.4 section 7.1.12.1

MLME-SET.request (MSTR)

MSTR requests to set certain MAC and PHY attributes.

Command format		***+MSTR=****{ varies according to attribute }*
Attribute	1 byte	Attribute # to be set (see table below)
Attribute	*See below*	Attribute value
		IEEE 802.15.4 section 7.1.13.1

The following PIB attributes are implemented. Caution should be used in setting attributes and changes may or may not have an effect depending on when they are made.

Attribute	Bytes	Attr #	Settable?
phyCurrentChannel	1 byte	0x00	Yes
phyTransmitPower	1 byte†	0x02	Yes
phyCCAMode	1 byte	0x03	Constant = 3
macAckWaitDuration	3 bytes	0x40	Constant (one second)
macAssociationPermit	1 byte (Boolean)	0x41	Yes
macBattLifeExt	1 byte (Boolean)	0x43	Constant = 0
macBattLifeExtPeriods	1 byte (Boolean)	0x44	Yes
macBeaconPayload	*macBeaconPayloadLength*	0x45	Yes
macBeaconPayloadLength	1 byte	0x46	Constant = 3
macBeaconOrder	1 byte	0x47	Constant = 15
macCoordExtendedAddress	8 bytes	0x4A	Yes
macCoordShortAddress	2 bytes	0x4B	Yes
macDSN	1 byte	0x4C	Yes
macMaxCSMABackoffs	1 byte	0x4E	Yes
macMinBE	1 byte	0x4F	Yes
macPANId	2 bytes	0x50	Yes
macPromiscuousMode§	1 byte (Boolean)	0x51	Yes
macShortAddress	2 bytes	0x53	Yes
macSuperframeOrder	1 byte	0x54	Constant = 15
macTransactionPersistenceTime	1 byte	0x55	Yes
All other attributes	Not implemented		

† Format as specified in CC2420 PA_LEVEL specification, e.g. FF = 0dBm, EF = -7dBm, etc.

§ In promiscuous mode, no address filtering nor MAC-level processing of received packets takes place. Received packets are translated to PDAI commands instead. This is intended for IEEE 802.15.4 sniffers. In this mode, MAC level requests will not function correctly.

IEEE 802.15.4 section 7.4 (Tables 71 and 72)

MLME-SET.confirm (MSTC)

MSTC confirms a request to set attribute data.

Command format		***+MSTC=****{2 bytes}*
status	1 byte	Result as enumeration
Attribute	1 byte	Attribute

IEEE 802.15.4 section 7.1.13.2

MLME-START.request (MSRR)

MSRR requests that a start operation is performed. Since beacon networks are not supported, the request affects coordinators only and its only effect is the set the logical frequency channel and the PAN ID. (For other devices, these parameters are set during orphan scan or association.)

Command format		***+MSRR=****{14 bytes}*
Flags	1 byte	Bit 0: PANCoordinator Bit 1: BatteryLifeExtension, should equal 0 Bit 2: CoordRealignment Bit 3: SecurityEnable
Filler1	3 bytes	*(fill with 00)*
PANid	2 bytes	PAN ID, if coordinator
LogicalChannel	1 byte	Logical frequency channel 0x0B – 0x1A, if coordinator
Filler1	5 bytes	*(fill with 00)*
BeaconOrder	1 byte	Beacon order (should equal 0F)
SuperframeOrder	1 byte	Superframe order (should equal 0F)
		IEEE 802.15.4 section 7.1.14.1

MLME-START.confirm (MSRC)

MSRC confirms the result of a start operation.

Command format		***+MSRC=****{varies according to attribute}*
status	1 byte	Result as enumeration
		IEEE 802.15.4 section 7.1.14.2

MLME-SYNC.request (MSYR)

MSYR requests to synchronize with a coordinator. The command relates to beacon networks and is currently not supported.

MLME-SYNC-LOSS.indication (MSLI)

MSLI indicates loss of synchronization with a coordinator. The command relates to beacon networks and is currently not supported.

MLME-POLL.request (MPLR)

MPLR requests data from the coordinator. These must be sent regularly by sleepy end devices in order to retrieve data that is being cached for them by a coordinator.

Note that the source address mode for the data request must match the addressing mode of the message being sent; the source address mode will be long if the short address is 0xFFFF or 0xFFFE, or short otherwise. An +MPLC confirmation will only be issued when no further data is pending; otherwise, the response will be an +MDAI data indication.

Command format		***+MRPL=****{26 bytes}*
Filler1	1 byte	*(fill with 00)*
SecurityEnable	1 byte	Security enabled
Filler2	5 bytes	*(fill with 00)*
CoordAddrMode	1 byte	Coordinator addressing mode
Filler3	8 bytes	*(fill with 00)*
DstAddr	8 bytes	Coordinator address. (If *DstAddrMode* specifies short addresses, ignore last 6 bytes.)
DstPanId	2 bytes	Coordinator PAN ID
		IEEE 802.15.4 section 7.1.16.1

MLME-POLL.confirm (MPLC)

MPLC confirms a request for data from the coordinator. If data is available, the response will be an MDAI data indication rather than an MPLC poll confirm.

Command format		***+MCPL=****{1 byte}*
status	1 byte	Result as enumeration. (Usually 0xEB = no more data)
		IEEE 802.15.4 section 7.1.16.2

Reference

Radio Frequency

Max RF output power	1mW = 0dBm
RF frequency range	2400MHz to 2485MHz
Communications protocol	IEEE 802.15.4 (DSSS O-QPSK chip encoding)
Raw data rate	250kbit/s
RF channels	16
Free space range with integral antenna	Approx 120m

Electrical

USB specification version	2.0
USB driver	`usbser.sys` (Supplied with Microsoft Windows)
USB plug	Type A
Maximum current draw	30mA

Mechanical

Max operating/storage temperature	-40ºC to +85 ºC
Dimensions L×W×H mm	55.7 × 20.3 x 10.5 (note 1)

1. Excludes USB plug

Regulatory

FCC compliance	G-antenna version compliant, awaiting certificate
CE compliance	G-antenna version compliant, awaiting certificate
IC (Industry Canada) compliance	G-antenna version compliant, awaiting certificate

Contact Information

FlexiPanel

Developed by:

FlexiPanel Ltd
2 Marshall St, 3rd Floor,
London W1F 9BB, United Kingdom
email: support@flexipanel.com
www.flexipanel.com

Manufactured and distributed by:

R F Solutions Ltd
Unit 21, Cliffe Industrial Estate,
Lewes, BN8 6JL, United Kingdom
email : sales@rfsolutions.co.uk
http://www.rfsolutions.co.uk
Tel: +44 (0)1273 898 000