PixieMAC ™

IEEE802.15.4 Transceiver with AT-style Command Interface

Summary

PixieMAC provides a serial command interface for the IEEE 802.15.4 standard for single-hop low data-rate networks. It is ideal for OEMs who need to add single-hop wireless serial communications to their products. It incorporates an FCC / CE certified IEEE 802.15.4 transceiver.

PixieMAC provides a command-oriented gateway between the IEEE 802.15.4 MAC and PHY layers and the host device. The StarLite firmware provides a less flexible but command-free interface for transparent cable replacement applications.

PixieMAC may be used freely with FlexiPanel Pixie and UZBee+ products. Additionally, in quantities of 2500 or more, it may be purchased as design licenses for integration directly on your main PCB.

Firmware Features:

- *Supports IEEE 802.15.4 non-beacon networks.*
- *Up to 32 cached messages (storage permitting).*
- *Certain devices may sleep*
- *Certain devices may broadcast*

Compatible Products

- *Fully compatible with other IEEE 802.15.4 products, including:*
  - *MACdongle*
  - *StarLite*
  - *StarLite USB*
- *FlexiPanel's ZigBee Demo Board may be used for evaluation*

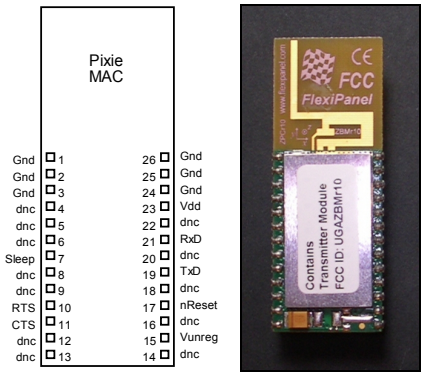


Fig 1. PixieMAC
(viewed from above)

Hardware Features:

- *2.4GHz IEEE 802.15.4 RF module*
- *FCC / CE / IC compliant*
- *Signature 'G' antenna, free-space range 120m, compact, low 'hand-effect' design*
- *115200 baud serial interface with flow control*
- *Sleep control input*
- *54mm x 20mm x 3mm*

Firmware Codes

Table 1. Firmware Codes	
Part No	***Description***
PIXIE-PXMC	PixieMAC (coordinator)
PIXIE-PXMF	PixieMAC (fast end device)
PIXIE-PXMS	PixieMAC (sleepy end device)

Manufactured to ISO9001:2000

Contents

Pin Connections

Pin Number	*Pin Name*	*Description*
1,2,3	*Gnd*	Power supply ground
4	*dnc*	Do not connect
5	*dnc*	Do not connect
6	*dnc*	Do not connect
7	*Sleep*	Sleep control; sleeps when high
8	*dnc*	Do not connect
9	*dnc*	Do not connect (note 3)
10	*RTS*	Flow control output / device ready indicator. Data may be sent to the RxD pin if RTS is low.
11	*CTS*	Flow control input (note 4). To suspend data transmission on the TxD pin, set CTS high.
12	*dnc*	Do not connect
13	*dnc*	Do not connect
14	*dnc*	Do not connect
15	*Vunreg*	Unregulated voltage input (note 2)
16	*dnc*	Do not connect
17	*nReset*	Reset input (active low) (note 1)
18	*dnc*	Do not connect
19	*TxD*	Serial output (8N1, 115200 baud)
20	*dnc*	Do not connect
21	*RxD*	Serial data input (8N1, 115200 baud)
22	*dnc*	Do not connect
23	*Vdd*	Regulated power supply input Regulated power supply output (note 2)
24,25,26	*Gnd*	Power supply ground

Table 1. Pin descriptions for PixieMAC

1. Should be pulled high via a 10K resistor for normal operation.
2. Requires optional voltage regulator option to be fitted for onboard regulation to be functional.
3. In future devices, this pin may become a Vddcore power supply pin. Refer to the *Future releases of Pixie & Pixie Lite* section of this data sheet and the documentation for 18F46L10 and 18F45L10 series devices from Microchip Technology (www.microchip.com).
4. Connect low if flow control is not required

IEEE 802.15.4 Overview

The IEEE 802.15.4 communications protocol is a low power, low data rate communications protocol. (practically speaking, approximately 38.4 kbaud in FlexiPanel products).

Communication is single-hop between up to 65K devices. Messages may be broadcast and any node can address any other.

Devices are allowed to sleep, but in doing so they must rely on the central coordinator to cache messages for them, and their ability to participate in communications with other devices will be limited.

The protocol is low cost and easy to implement. It is the protocol of choice if greater complexity is not required.

No profiles are defined by the IEEE 802.15.4 standard. Data is simply transferred as a payload in a packet which may be up to 127 bytes, including packet addressing headers.

Each IEEE 802.15.4 device must be assigned a unique MAC address. IEEE 802.15.4 products from FlexiPanel are pre-assigned MAC addresses. In some instances it is not possible to store a MAC address on the product when shipped and you will need to request an allocation of MAC addresses from us.

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

PixieMAC Overview

PixieMAC 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 PixieMAC 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 PixieMAC.

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

Device Programming

Pixie products are available from distributors pre-loaded with PixieMAC firmware. Alternatively it may be loaded onto any Pixie module using a PIC programmer such as the low-cost TEAclipper. The firmware may be downloaded from *www.hexwax.com*.

Design Licensing

Design licensing allows you to integrate the Pixie design directly on your main PCB rather than as a separate module. This delivers exceptional cost savings in terms of labor and parts, and offers you much greater control over the supply chain.

We require an up-front purchase of a minimum quantity of 2500 encrypted PixieMAC *.wax* firmware licenses, and signature of a confidentiality agreement. In return we will give you a pack containing Gerber CAD files and bills of materials. Only the encrypted version of the firmware is compatible with the design supplied.

Contact *support@flexipanel.com* for more information.

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.

- **`PXMC:`** PixieMAC Coordinator, creates a IEEE 802.15.4 network
- **`PXMF:`** PixieMAC Full Function Device, Rx-On-When-Idle, participates in a IEEE 802.15.4 network, cannot sleep
- **`PXMS:`** PixieMAC Reduced Function Device, Rx-Of-When-Idle, participates in a IEEE 802.15.4 network, can sleep

Frequency channels

Pixie 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 PXMS versions of the firmware are allowed to sleep. They are put into sleep mode when the sleep pin goes high; during sleep, the RTS pin will output high.

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.

Copy Protection

To protect against copying, if the PixieMAC firmware is run on any hardware except FlexiPanel Pixie products, it will cease to function after approximately two minutes. Steinlaus tags are also included in the code.

Evaluation Kit

The easiest way to get to know PixieMAC is with the ZigBee Evaluation Kit available from FlexiPanel. This will also require a Microchip ICD2 In-Circuit Debugger to program the firmware into the Pixie / Pixie Lite supplied.

In the evaluation boards, the I/O pins are connected as follows:

Pin Number	*Pin Name*	*Description*
7	*Sleep*	Switch labeled "EP2 A2"
10	*RTS*	LED labeled "A4 / EP5 / RTS"
11	*CTS*	Switch labeled "Config SW" *Ensure jumper A8 – A9 is fitted.*
17	*nReset*	Reset push switch
19	*TxD*	Serial output (8N1, 115200 baud)
21	*RxD*	Serial data input (8N1, 115200 baud)

Please note the following:

1. Remove A1-B1, A2-B2, A3-B3 during programming and fit them again after. The configuration bits are specified in the file "FCS PixieMAC".

2. For RS232 connection, fit jumper A4-B4. For Pixie Config Tool connection, remove the jumper. (Applies to ZEVr4 and higher board revisions.)

3. Fit jumpers A5-B5, A6-B6, A10-B10.

4. Fit jumper A8-A9. This connects the CTS input to the switch labeled "Config SW" so you can simulate flow control being halted by the host device. *In normal operation, it must be in the low position or you will not get a response from the PixieMAC!*

5. The sleep input connects to the switch labeled "EP2". Normally this should be in the low position. If you put it in the low position, Pixie will enter its sleep mode. This will be indicated by the RTS line going high and the RTS led lighting. (Applies to PXMS firmware only; the others can't sleep.)

Source code

This firmware is a simple command-based wrapper for the MAC API IEEE 802.15.4 library. It may not suit all customers; source code is available for those who wish to modify it. Contact FlexiPanel for details.

Release notes, version 0B40011310362105110 6tt

tt refers to the device type. Refer to the **DVRC** message for details.

In this release, security is not supported. SQTP programming of MAC addresses is supported – refer to the Pixie data sheet for details. The external oscillator is used, so the 16MHz crystal must be fitted.

Bibliography

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

DS500, RF Transceiver Selection Guide downloadable from *www.FlexiPanel.com*.

Usage examples

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

Application software interfacing with PixieMAC 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.

Presence Detection

The presence of PixieMAC can be checked by issuing a version number request:

+DVRR	(request version)
+DVRC=0B400113103621051106 11	(confirmation – version number may vary)

Setting 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.

Disabling Timeouts

If you're evaluating PixieMAC 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=04000101FFFF0303989898989898989834343434343434FFFF8890abcdef
(request send data 90ABCDEF, length 04 from 9898989898989898 to 3434343434343434, long addr mode (0303))

+MDAC=00000101FFFF0303989898989898989834343434343434FFFF5F
(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. During initialization, 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=04001601FFFF0303343434343434343498989898989898FFFF19E890ABCDEF
(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=00000000000001080 0F8FF07 (request active scan, all channels 0x0B – 0x1A)
+MSCC=EA00000000000108FF0700F8 (confirmation – scan result, nothing found)

Another example:

+MSCR=000000000000010800F8FF07 (request scan)
+MBNI=03000C01940D020B0000FF07B98084B043D3120110435A00FF4F44E0001100
(indication – response from network 940D)
+MSCC=00011201120101 0BFF0700F8000B940D0000FF07B98084B0FF4F43D31201E00000
(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=8E001401FFFF0302343434343434343400004 0BDE4306B2C3412E9
(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=00000000000078560000000000000000343434343434343 4
(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=8E00000000000B02000000000000000000000000000000003412

(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

(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=000000000000303FF070000 (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

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

+MDAC=00000100341202020000000000000000010000000000000341253

(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

(request send data 90ABCDEF from Addr 0000 / PAN 1234 to Addr 0001 / PAN 1234, indirect, Acknowledged (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=00000100341202020000000000000000010000000000000341253

(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=040001003412020200000000000000000FFFF00000000000034128890abcdef

(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

For non-sleepy devices, received packets generate a MAC-level indication message, *e.g.*:

+MDAI=04500A013412020200009898989898980100343434343434341225E890ABCDEF
(indication –packet received)

If the device is a sleepy device, it must explicitly 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 PixieMAC can be used as a sniffer detector.

Message Reference

Format

Messages sent to and received from PixieMAC 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, PixieMAC 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, PixieMAC 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 syntax incorrect
02	Command not supported
03	MAC address has already been set, cannot be changed
04	MAC address not valid – send **+DRMC=** message first

Enquire MAC Address request (DMCR)

DMCR enquires the MAC address of PixieMAC. PixieMAC 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 PixieMAC. 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***. If the MAC address ahs not yet been set, the value will be all zeroes.

Set MAC Address request (DSMR)

DSMR specifies a MAC address for PixieMAC. 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 PixieMAC 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. PixieMAC 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
```

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*.

Version request (DVRR)

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

```
+DVRR
```

Version confirm (DVRC)

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

```
+DVRC=0B400113103621051106 11
```

The response takes the form ***+DVRC=PPPPPPPPVVVVVVDDMMYYTT***, where the digits are as follows:

PPPPPPPP	Product ID (0B400113 indicates PixieMAC firmware)
VVVVVV	Pixie MAC version number
DDMMYY	Firmware release date
TT	Device type

TT = 01	PixieMAC coordinator on Pixie
TT = 02	PixieMAC fast end device on Pixie
TT = 03	PixieMAC sleepy end device on Pixie
TT = 04	PixieMAC coordinator on Pixie
TT = 05	PixieMAC fast end device on Pixie
TT = 06	PixieMAC sleepy end device on Pixie

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
		IEEE 802.15.4 section 6.2.1.3

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: Bit 0 = acknowledged transmission Bit 1 = GTS transmission Bit 2 = indirect transmission Bit 3 = security enabled transmission.
SrcPanId	2 bytes	Source PAN ID
SrcAddrMode	1 byte	Source addressing mode (0x02 = short, 0x03 = long)
DstAddrMode	1 byte	Destination addressing mode (0x02 = short, 0x03 = long)
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. (Ignored; the current *macDSN* value is used and then incremented.)
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 (0x02 = short, 0x03 = long)
DstAddrMode	1 byte	Destination addressing mode (0x02 = short, 0x03 = long)
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 0x00 = Association successful 0x01 = PAN at capacity 0x02 = PAN access denied
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	Short address allocated to this device by the coordinator
		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 beacon.

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 (0x02 = short, 0x03 = long)
LogicalChannel	1 byte	Logical channel
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 (*AssocPermit* is bit 15)
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 non-volatile 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 channel
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 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)
AttributeValue	*See below*	Attribute value
		IEEE 802.15.4 section 7.1.13.1

The following attributes are implemented. Caution should be used in setting attributes. Setting values may or may not have an effect depending on the state of the stack.

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.			

‡ Nonvolatile value, will be remembered after power off § 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. MAC level requests may fail.
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
LogicalChannel	1 byte	Logical frequency channel
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.

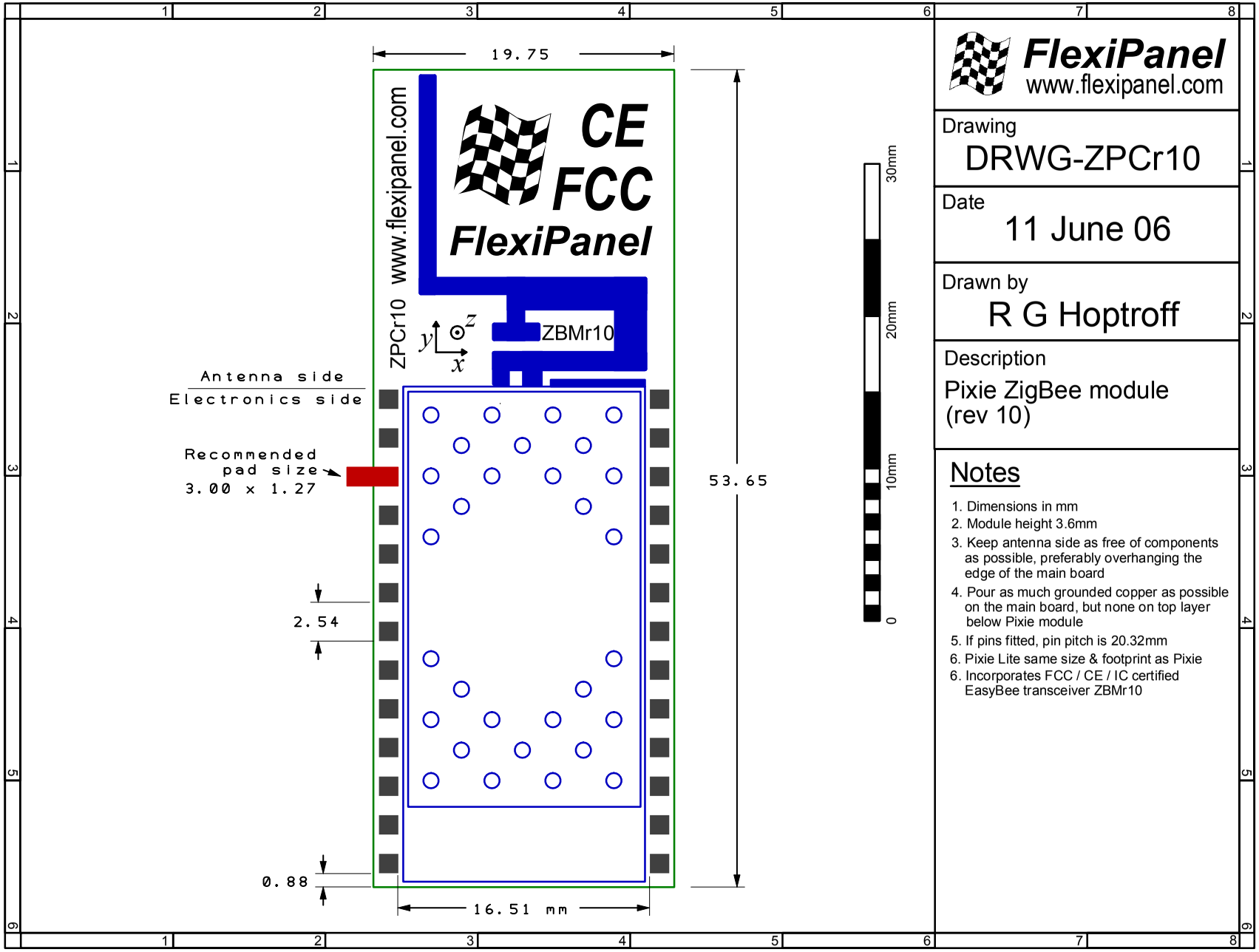
Note that the source address mode for the data request must match the addressing mode of the message being sent; the source address mode will be long if the short address is 0xFFFF or 0xFFFE, or short otherwise. An +MPLC confirmation will only be issued when no further data is pending; otherwise, the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

Command format		***+MPLR=****{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 (0x02 = short, 0x03 = long)
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



FlexiPanel
www.flexipanel.com

Drawing
DRWG-ZPCr10

Date
11 June 06

Drawn by
R G Hoptroff

Description
Pixie ZigBee module
(rev 10)

Notes

1. Dimensions in mm
2. Module height 3.6mm
3. Keep antenna side as free of components as possible, preferably overhanging the edge of the main board
4. Pour as much grounded copper as possible on the main board, but none on top layer below Pixie module
5. If pins fitted, pin pitch is 20.32mm
6. Pixie Lite same size & footprint as Pixie
6. Incorporates FCC / CE / IC certified EasyBee transceiver ZBMr10

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

Current consumption, excluding I/O pins	≤30mA
Current consumption, sleep mode	2µA est
Supply Voltage (regulated) Vcc	2.1V to 3.6V

Mechanical

Max operating/storage temperature	−40ºC to +85 ºC
Dimensions L×W×H mm	54 × 20 x 3 (excluding legs in DIL version)

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