Summary

The MAC API is designed to be a very simple programming interface for developers of non-beacon IEEE 802.15.4 applications. It may be used for the rapid application development of single-hop products with minimal prior knowledge of the IEEE 802.15.4 specification or radio communications.

IEEE 802.15.4 is designed for single-hop low-power, low data rate networks. Its multi-hop extension. known as ZigBee, is more well-known. However, the multi-hop nature of ZigBee means that it is more complicated to implement. In addition, broadcasting to all nodes in the network is much more efficient in a single-hop network, making it more suited to data-bus (e.g. RS485) cable replacement.

Certain IEEE 802.15.4 devices may sleep. However, in order to do so, their messages must be cached by the network coordinator and their ability to participate in direct communications with other devices in the network will be limited.

Compatibility

The API is fully compatible with all non-beacon IEEE 802.15.4 products, including:

- *StarLite*
- *StarLite USB*
- *MAC API*
- *MACdongle*

Features

MAC API incorporates the following features:

- Long- and short-address data communications
- Association and disassociation
- Active, passive energy density and orphan scans
- Device sleep and coordinator polling
- Orphan indication
- Promiscuous mode for packet sniffing

MAC API is free, provided it is used with FlexiPanel Pixie radio products only.

Contents

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*.

MAC API Overview

MAC API 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 MAC API devices and other devices in the FlexiPanel IEEE 802.15.4 firmware range. The Usage Examples section that follows shows examples of actual function calls. In addition, the source code for our two commercial applications that use the MAC API (StarLite and MAC API) is available for inspection in the MAC API development kit.

Device Types

Three types of device are implemented on each of Pixie and Pixie Lite, each with a different library build. It is not possible to switch between device types at runtime:

`MAC API PixC.lib`	Pixie Coordinator
`MAC API PixF.lib`	Pixie Fast End Device
`MAC API PixS.lib`	Pixie Sleepy End Device
`MAC API PixLiteC.lib`	Pixie Lite Coordinator
`MAC API PixLiteF.lib`	Pixie Lite Fast End Device
`MAC API PixLiteS.lib`	Pixie Lite Sleepy End Device

Coordinators, usually one per network, decide network parameters such as operating frequency, network membership and short address. They should remain on all the time and are able to cache messages for devices that sleep.

End devices participate in a network created by a coordinator. Fast end devices stay on all the time, can send messages to any other devices and can transmit and receive broadcast messages. Sleepy end devices use the coordinator to cache message for them. This allows them to sleep, at the cost of only being able to communicate with the coordinator and unable to accept broadcast communications while sleeping.

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.

Releasing Memory

It is very important that you free the memory passed back to you by MLME_BEACON_NOTIFY_indication, MCPS_DATA_indication, PD_DATA_indication and MLME_SCAN_confirm, even if you do not anticipate using these functions.

Copy Protection

To protect against copying, if the MAC API 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 MAC API 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 Pixies 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 MAC API".

2. For RS232 connection, fir 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 MAC API!*

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.)

Release notes, version 0B40011510362105110 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.

Bibliography

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

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

Firmware Development Guide

The Microchip Technology MPLAB development environment and C18 compiler will be needed to develop MAC API applications. A debugger such as the Microchip Technology ICD2 is recommended. Please refer to Microchip Technology documentation for full details on how to develop applications for PIC microprocessors.

A MailBox application project include the following files:

`MailBox.h`	Header file for MailBox library functions and data.
`MailBoxAPI-`*`HSSD`*`.lib`	MailBox library. *HSSD* signifies the different library versions: *H* indicates hardware, being Pixie (H) or Pixie Lite (L). *SS* indicates stack profile, for example Home Controls (HC). *D* indicates device type, being Coordinator (C), Router (R), Fast End Device (F) or Sleepy End Device (S).
`MbxLink`*`xxxx`*`.lkr`	Required linker script. *XXXX* is 4620 for Pixie and 2520 for Pixie Lite.

You will also need to provide code for your main application program and also specify the configuration bits you require. The following memory model settings should be specified:

- Small code model
- Large data model
- Single-bank model

The oscillator configuration must be set for a 16MHz clock. If using the internal oscillator block, set the oscillator setting to Internal RC and include the following lines in your startup code:

```
OSCCONbits.IRCF1 = 1;      // changes to <IDCF2:IDCF0>=110 = 4MHz
OSCTUNEbits.PLLEN = 1;     // PLL 4MHz -> 16MHz
Delay1KTCYx( 100 );        // allow 25ms for clock to settle
```

Applications Examples

The firmware for the StarLite application is included in the developer's kit. Please study it as an example of implementing an application using MAC API.

The PixieMAC application is a wrapper to the MAC API. It provides access to the MAC layer via AT-like commands. The *Usage Examples* section of the PixieMAC documentation contains extended information on how the various requests, confirmations, indications and responses work.

Function Reference

Data Types

A variety of data structures are declared in `MAC API.h`. Most have rather obvious functions and will not be documented in detail – refer to `MailBox.h` for further information.

The *MailBoxParams* structure is extensive and is detailed in the *MailBoxTasks()* section. For efficiency, it contains unused sections with variable names beginning with the word `filler`. These may be ignored.

MACAPIInit()

MACAPIInit() initializes the stack. It will enable high and low priority interrupts. The function should be called once during initialization.

MACAPITasks()

This section describes the function of the MAC API state machine based on its *currentPrimitive* argument on entry to and on exit from *MailBoxTasks*.

Related information is stored in the *params* structure. For example, the *SrcAddr* variable referred to in *MCPS_DATA_request* below is actually *params.MCPS_DATA_request. SrcAddr*.

All *params* fields named *Status* will be 0x00 to indicate success; otherwise the status code will be equal to an IEEE 802.15.4 status code (refer to IEEE 802.15.4 specification section 2.1).

There are four types of primitive: *Requests* made by the application, *Confirmations* of requests returned by the stack, asynchronous *Indications* provided to the application, and *Responses* which are required by the stack for some indications.

Unless otherwise noted, a new request should not be made unless a confirmation has been received pertaining to the previous request.

It is very important that you free the memory passed back to you by MLME_BEACON_NOTIFY_indication, MCPS_DATA_indication, PD_DATA_indication and MLME_SCAN_confirm, even if you do not anticipate using these functions.

PHY Messages

PD-DATA.indication

When promiscuous mode is not set, *PD_DATA_indication* should be treated as an internal MAC API state..

When promiscuous mode is set, (refer to MLME_GET_request), no address filtering is applied and all received packets are intercepted at the PHY level and passed to the application as *PD_DATA_indication* 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.

Note that the *psdu* payload must be freed by the application after use by calling *MACDiscardRx()*. When promiscuous mode is set, application **must** call *MACDiscardRx()* to free the *sdu* data payload once the *PD_DATA_indication* has been processed.

PD_DATA_indication	
psduLength	Packet length in bytes (PHR)
psdu	Pointer to PHY payload
	IEEE 802.15.4 section 6.2.1.3

Internal PHY States

The following states are always internal to the MAC API:

PD_DATA_request
PD_DATA_confirm
PLME_CCA_request
PLME_CCA_confirm
PLME_ED_request
PLME_ED_confirm
PLME_GET_request
PLME_GET_confirm
PLME_SET_TRX_STATE_request
PLME_SET_TRX_STATE_confirm
PLME_SET_request
PLME_SET_confirm

When the stack is in one of these states, the application should not modify the *currentPrimitive* or the *params* structure. When it has completed its own processing, the application should call MACAPITasks() again with these variables unaltered.

MAC Messages

MCPS-DATA.request

MCPS_DATA_request requests a data packet be transmitted. Prior to making the request, the payload data should be placed in the buffer *pTxData*. A MCPS_DATA_confirm of an indirect transmission will only be generated when the transmission has completed; in this case, you do not have to wait for the MCPS_DATA_confirm before issuing the next request.

MCPS_DATA_request	
msduLength	Length of payload *msdu* to follow
FrameType	0x00 = Beacon frame 0x01 = Data frame 0x02 = Ack frame 0x03 = MAC command frame
TxOptions	Transmit options: Bit 0 = acknowledged transmission Bit 1 = GTS transmission Bit 2 = indirect transmission Bit 3 = security enabled transmission.
SrcPanId	Source PAN ID
SrcAddrMode	Source addressing mode (0x02 = short, 0x03 = long)
DstAddrMode	Destination addressing mode (0x02 = short, 0x03 = long)
SrcAddr	Source address. (If *SrcAddrMode* specifies short addresses, ignore last 6 bytes.)
DstAddr	Destination address. (If *DstAddrMode* specifies short addresses, ignore last 6 bytes.)
DstPanId	Destination PAN ID
msduHandle	Data handle. (Ignored; the current *macDSN* value is used and then incremented.)
msdu	(Ignored)
	IEEE 802.15.4 section 7.1.1.1

MCPS-DATA.confirm

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

MCPS_DATA_confirm	
status	Result as enumeration
msduHandle	Data handle (equals macDSN number at time of request)
	IEEE 802.15.4 section 7.1.1.2

MCPS-DATA.indication

MCPS_DATA_indication 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.

The application **must** call *MACDiscardRx()* to free the *msdu* data payload once an MCPS_DATA_indication has been processed.

MCPS_DATA_indication	
msduLength	Length of payload *msdu*
SecurityUse	Security indicator
SrcPanId	Source PAN ID
SrcAddrMode	Source addressing mode (0x02 = short, 0x03 = long)
DstAddrMode	Destination addressing mode (0x02 = short, 0x03 = long)
SrcAddr	Source address. (If *SrcAddrMode* specifies short addresses, ignore last 6 bytes.)
DstAddr	Destination address. (If *DstAddrMode* specifies short addresses, ignore last 6 bytes.)
DstPanId	Destination PAN ID
mpduLinkQuality	Link quality
ACLEntry	*macSecurityMode* parameter
msdu	Payload received
	IEEE 802.15.4 section 7.1.1.3

MCPS-PURGE.request

MCPS_PURGE_request requests a transmit packet gets purged from the queue.

MCPS_PURGE_request	
msduHandle	Data handle
	IEEE 802.15.4 section 7.1.1.4

MCPS-PURGE.confirm

MCPS_PURGE_confirm reports on a purge operation.

MCPS_PURGE_confirm	
status	Result as enumeration
msduHandle	Data handle
	IEEE 802.15.4 section 7.1.1.5

MLME-ASSOCIATE.request

MLME_ASSOCIATE_request requests an association with a PAN coordinator.

MLME_ASSOCIATE_request	
CapabilityInfo	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	True if security enabled
LogicalChannel	Channel on which to associate
CoordAddrMode	Coordinator addressing mode (0x02 = short, 0x03 = long)
CoordAddress	Coordinator address. (If *CoordAddrMode* specifies short addresses, ignore last 6 bytes.)
CoordPANid	Coordinator PAN ID
	IEEE 802.15.4 section 7.1.3.1

MLME-ASSOCIATE.indication

MLME_ASSOCIATE_indication indicates the reception of an association request from another device. An MLME_ASSOCIATE_response ***must*** be generated in response to the indication.

MLME_ASSOCIATE_indication	
CapabilityInfo	Capabilities of associating device (see +*MASR*)
SecurityEnable	True if security enabled
DeviceAddr	Associating device address
ACLEntry	*macSecurityMode* parameter
	IEEE 802.15.4 section 7.1.3.2

MLME-ASSOCIATE.response

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

MLME_ASSOCIATE_response	
Status	Result as enumeration 0x00 = Association successful 0x01 = PAN at capacity 0x02 = PAN access denied
SecurityEnable	True if security enabled
AssocShortAddr	Short address allocated
DeviceAddress	Address of device requesting association
	IEEE 802.15.4 section 7.1.3.3

MLME-ASSOCIATE.confirm

MLME_ASSOCIATE_confirm confirms the completion of an association request.

MLME_ASSOCIATE_confirm	
Status	Result as enumeration
DeviceAddr	Short address allocated to this device by the coordinator
	IEEE 802.15.4 section 7.1.3,4

MLME-DISASSOCIATE.request

MLME_DISASSOCIATE_request requests disassociation from a PAN coordinator.

MLME_DISASSOCIATE_request	
DisassocReason	Dissociation reason
SecurityEnable	True if security enabled
DeviceAddress	Device address
	IEEE 802.15.4 section 7.1.4.1

MLME-DISASSOCIATE.indication

MLME_DISASSOCIATE_indication indicates the reception of a disassociation request.

MLME_DISASSOCIATE_indication	
DisassocReason	Dissociation reason
SecurityEnable	True if security enabled
DeviceAddress	Device address
ACLEntry	*macSecurityMode* parameter
	IEEE 802.15.4 section 7.1.4.2

MLME-DISASSOCIATE.confirm

MLME_DISASSOCIATE_confirm confirms the completion of a disassociation request.

MLME_DISASSOCIATE_confirm	
Status	Result as enumeration
	IEEE 802.15.4 section 7.1.4.3

MLME-BEACON-NOTIFY.indication

MLME_BEACON_NOTIFY_indication indicates the reception of a beacon. The application **must** call *MACDiscardRx()* to free the *sdu* data payload once the MLME_BEACON_NOTIFY_indication has been processed.

MLME_BEACON_NOTIFY_indication	
sduLength	Length of payload *sdu*
SecurityUse	Security indicator
CoordPanId	Coordinator PAN ID
CoordAddrMode	Coordinator addressing mode (0x02 = short, 0x03 = long)
LogicalChannel	Logical channel
CoordAddr	Coordinator address. (If *CoordAddrMode* specifies short addresses, ignore last 6 bytes.)
Timestamp	Timestamp
BSN	Sequence number
SecurityFailure	Security failure
GTSpermit	Coordinator accepts GTS requests
SuperframeSpec	Superframe specification (*AssocPermit* is bit 15)
LinkQuality	Link quality
ACLEntry	*macSecurityMode* parameter
(AddrList)	*(Beacon networks not supported, so AddrList not provided)*
sdu	Beacon payload
	IEEE 802.15.4 section 7.1.5.1

MLME-GET.request

MLME_GET_request requests MAC and PHY attribute data. Refer to MLME-SET for a list of available attributes. If the *currentPrimitive* is set to MLME_GET_request, MACAPITasks() is guaranteed to return immediately with *currentPrimitive* is set to MLME_GET_confirm. Therefore g et requests may be performed *ad hoc* rather than as part of the state machine loop, provided the state machine is idle.

MLME_GET_request	
Attribute	Attribute requested
	IEEE 802.15.4 section 7.1.6.1

MLME-GET.confirm

MLME_GET_confirm confirms attribute data.

MLME_GET_confirm	
status	Result as enumeration
Attribute	Attribute (see table in MLME_SET_confirm section)
AttributeValue	Attribute value
	IEEE 802.15.4 section 7.1.6.2

MLME-GTS.request

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

MLME-GTS.confirm

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

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

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

MLME_ORPHAN_indication	
SecurityUse	True if security enabled
OrphanAddr	Orphan device address
ACLEntry	*macSecurityMode* parameter
	IEEE 802.15.4 section 7.1.8.1

MLME-ORPHAN.response

MLME_ORPHAN_response responds to the presence of an orphaned device.

MLME_ORPHAN_response	
AssociateMember	True if associated with this coordinator
SecurityEnable	True if security enabled
OrphanAddr	Orphan address
ShortAddr	Short address
	IEEE 802.15.4 section 7.1.8.2

MLME-RESET.request

MLME_RESET_request requests that a reset operation is performed.

MLME_RESET_request	
SetDefaultPIB	If true, resets non-volatile attributes
	IEEE 802.15.4 section 7.1.9.1

MLME-RESET.confirm

MLME_RESET_confirm confirms the result of a reset operation.

MLME_RESET_confirm	
status	Result as enumeration
	IEEE 802.15.4 section 7.1.9.2

MLME-RX-ENABLE.request

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 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

MLME_SCAN_request 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.

MLME_SCAN_request	
Scan type	Scan type (00 = Energy detect, 01 = Active scan, 02 = Passive scan, 03 = Orphan scan
Scan duration	Scan duration
Scan channels	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

MLME_SCAN_confirm 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.

Note: all scan confirmations except orphan scan return lists that must be freed from memory using *SRAMfree()* after use. Passive and Active scans return a linked list of memory blocks, all of which must be freed.

MLME_SCAN_confirm	
status	Result as enumeration (0xEA = No networks found on active scan)
ResultListSize	Number of results returned
Filler1	*(ignore)*
Scan type	Scan type
Filler2	*(ignore)*
Unscanned channels	Unscanned channels
EnergyDetectList†	Energy detect list (1-byte values equal to [RSSI+128])
PanDescrList†	PAN Descriptor list (20-byte PAN Descriptor values)
† EnergyDetectList for energy detect scans only. *PanDescrList* is for active and passive scans only. If the pointer is to an *EnergyDetectList*, it is a single block of memory that must be freed with *SRAMfree()*. If the pointer is to a *PanDescrList*, it is a linked list of blocks each of which must be freed with *SRAMfree()*.	
	IEEE 802.15.4 section 7.1.11.2

The PAN Descriptor List elements have the following format:

Flags	Bit 0: 1 for *CoordAddrMode* long, 0 for short Bit 1: GTSPermit Bit 2: SecurityUse Bit 3: SecurityFailure Bits 4-7: ACLEntry
LogicalChannel	Logical channel
CoordPANid	Coordinator PAN ID
CoordAddress	Coordinator address. (If *CoordAddrMode* specifies short addresses, ignore last 6 bytes.)
SuperframeSpec	Superframe specification
Timestamp	Timestamp
LinkQuality	Link quality as reported by CC2420
next	Pointer link to next element in PAN Descriptor List
	IEEE 802.15.4 section 7.1.11.2

MLME-COMM-STATUS.indication

MLME_COMM_STATUS_indication 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.

MLME_COMM_STATUS_indication	
status	Result as enumeration
PanId	PAN ID (not populated since always macPANId)
SrcAddrMode	Source addressing mode (not populated since always 0x03=long)
DstAddrMode	Destination addressing mode (not populated since always 0x03)
SrcAddr	Source address (not populated since always MAC address)
DstAddr	Destination address.
	IEEE 802.15.4 section 7.1.12.1

MLME-SET.request

MLME_SET_request requests to set certain MAC and PHY attributes. If the *currentPrimitive* is set to MLME_SET_request, MACAPITasks() is guaranteed to return immediately with *currentPrimitive* is set to MLME_SET_confirm. Therefore set requests may be performed *ad hoc* rather than as part of the state machine loop, provided the state machine is idle.

MLME_SET_request	
Attribute	Attribute # to be set (see table below)
AttributeValue	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 PIBs	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

MLME_SET_confirm confirms a request to set attribute data.

MLME_SET_confirm	
status	Result as enumeration
Attribute	Attribute
	IEEE 802.15.4 section 7.1.13.2

MLME-START.request

MLME_START_request requests that a start operation is performed. For non-beacon networks, this doesn't amount to much: for coordinators, it sets the frequency channel and PAN ID, but not the short address.

MLME_START_request	
Flags	Bit 0: PANCoordinator Bit 1: BatteryLifeExtension, should equal 0 Bit 2: CoordRealignment Bit 3: SecurityEnable
PANid	PAN ID
LogicalChannel	Logical frequency channel
BeaconOrder	Beacon order (should equal 0F)
SuperframeOrder	Superframe order (should equal 0F)
	IEEE 802.15.4 section 7.1.14.1

MLME-START.confirm

MLME_START_confirm confirms the result of a start operation.

MLME_START_confirm	
status	Result as enumeration
	IEEE 802.15.4 section 7.1.14.2

MLME-SYNC.request

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

MLME-SYNC-LOSS.indication

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

MLME-POLL.request

MLME_POLL_request 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 that would have originated the message 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 MLME_POLL_confirm confirmation will only be issued when no further data is pending; otherwise, the response will be an MLME_DATA_indication.

MLME_POLL_request	
SecurityEnable	Security enabled
CoordAddrMode	Coordinator addressing mode (0x02 = short, 0x03 = long)
DstAddr	Coordinator address. (If *DstAddrMode* specifies short addresses, ignore last 6 bytes.)
DstPanId	Coordinator PAN ID
	IEEE 802.15.4 section 7.1.16.1

MLME-POLL.confirm

MLME_POLL_confirm 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.

MLME_POLL_confirm	
status	Result as enumeration
	IEEE 802.15.4 section 7.1.16.2

Callback Functions

The following three callback functions must be provided even if they are not used:

void PriorityUserInterruptHandler(void)

Used for very high priority interrupt processing. This takes precedent over all other processing, including caching incoming ZigBee messages. Anything using the priority interrupt handles must complete very quickly. Typically it will be used to note that an event has occurred, for example UART data has been received, so that it may be processed at a later time.

void UserInterruptHandler(void)

Used for normal priority interrupt processing.

void MACAPIHook (void)

ZigBeeHook allows advanced users to sneak look at the state of the ZigBee stack. If you investigate the ZigBee stack source, you can declare variables as extern and inspect them from within this hook function. DO NOT modify any variables - that would invalidate stack compliance.

Utility Functions

The utility functions are parts of the ZigBee stack which have been exposed because the application may also have independent uses for them. *Note: MACAPIInit() must be called before using any of these functions.*

void NVMWrite(NVM_ADDR *dest, BYTE *src, BYTE count)

NVMWrite writes up to 255 bytes to nonvolatile memory. Note that the entire 0x40-byte-aligned section of memory is erased and restored during this process, so the entire section of memory should not contain executable code. Data is verified before writing, so writing identical values to the memory does not exhaust it.

void NVMRead(BYTE *dest, NVM_ADDR *src, BYTE count)

NVMRead reads up to 255 bytes of nonvolatile memory.

TICK TickGet(void)

TickGet provides a clock which counts symbols (1/62,500ths of a second). The timer is 4 bytes, so the clock rolls over very 19 hours.

TICK TickGetDiff(TICK a, TICK b)

TickGetDiff calculates the difference between two time values.

unsigned char * SRAMalloc(unsigned char nBytes)

SRAMalloc allocates and returns the address of nBytes of RAM from the heap. Zero is returned if the memory could not be allocated. (*MACDiscardRx()* should be used for data payloads returned from *PD_DATA_indication*, *MCPS_DATA_indication*, *MLME_BEACON_NOTIFY_indication*.)

void SRAMfree(unsigned char * pSRAM)

SRAMfree must be used to free memory previously allocated with SRAMalloc. This includes blocks allocated by the stack by MLME_SCAN_confirm, which must be freed by the application.

Macros

Note that setting nonvolatile MIB values may not necessarily have any effect until the ZigBee stack is next initialized. *MACAPIInit() must be called before using any of these functions.*

pTxData is a buffer where the data payload should be stored prior to a *MCPS_DATA_request*.

IsMACReady() must return zero prior to filling ***pTxData*** with data; if it does not, continue to call MACAPITasks() until it does.

MACBlockTx() should be called once IsMACReady() returns zero in order to reserve the ***pTxData***.

The application must call ***MACDiscardRx()*** to free data payloads returned from *PD_DATA_indication*, *MCPS_DATA_indication*, *MLME_BEACON_NOTIFY_indication*

SetMACAddress(x), ***GetMACAddress(x)*** set and retrieve ***MIB_MACaddress***, where ***x*** is a pointer to an 8-byte buffer in RAM. The device must be reset after setting this value.

IsMACAddressValid() is nonzero if the MAC address has been set, either using the ***SetMACAddress()*** macro or SQTP programming.

TickGet() returns a four-byte timer value counting 1/62500ths of a second. .

TickGetDiff() returns the difference values returned by *TickGet()* . .

RANDOM_LSB, ***RANDOM_MSB*** provide a pseudorandom number generated from the Timer0 clock.

Development Kit Inventory

The MAP API Development Kit contains:

1. The files required for MPLAB-based applications development:

   ```
   MAC API.h
   MAC API PixC.lib
   MAC API PixF.lib
   MAC API PixS.lib
   MAC API PixLiteC.lib
   MAC API PixLiteF.lib
   MAC API PixLiteS.lib
   MACLink4620.lkr
   MACLink2520.lkr
   ```

2. This documentation, `MAC API DS501.pdf`.

3. The files `StarLite.c`, and `Config.c`, which are required for the StarLite example application, and project files for the six different builds PXMC, PXMF, and PXMS.

4. The StarLite documentation, `StarLite DS505.pdf`.

The MPLAB development environment and C18 compiler must be bought separately from Microchip Technology Inc (www.microchip.com) or one of its distributors.

Development Support

FlexiPanel Ltd publishes free Sniffer firmware which parses MAC API frames which may prove useful. Please contact us if you have any comments or suggestions.

Revision History

Version	Date	Major revisions
10-36-21	05-11-06	Initial release

Please note the following with release 103621051106:

- Security not implemented.

Contact details

FlexiPanel

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