802.15.4 MAC
Application Programming Interface

Version: 1.1

Document Number: F8W-2005-1503

March 19, 2007

Texas Instruments
San Diego, CA USA
+1 (619) 542-1200

TABLE OF CONTENTS

1. Introduction

1.1 Purpose

This document describes the application programming interface for the 802.15.4 MAC software. The API provides an interface to the management and data services of the 802.15.4 stack.

1.2 References

1. IEEE Std 802.15.4-2006, *Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)*, September 8, 2006.
2. F8W-2003-0002, *Z-Stack OS Abstraction Layer Application Programming Interface*, version 1.4, Texas Instruments, November 27, 2006.

1.3 Acronyms and Definitions

AES	Advanced Encryption Standard encryption algorithm.
API	Application Programming Interface.
CAP	Contention Access Period
CBC-MAC	Cipher Block Chaining Message Authentication Code.
CCM	Encryption using CTR with CBC-MAC.
Coordinator	A full function device that accepts associations and transmits beacons.
CSMA-CA	Carrier Sense Multiple Access with Collision Avoidance.
CTR	Counter mode encryption.
DSN	Data Sequence Number
FFD	Full Function Device.
LQI	Link Quality Indication.
MAC	Medium Access Control.
MCPS	MAC Common Part Sublayer (data interface).
MIC	Message integrity code.
MLME	MAC Sublayer Management Entity (management interface).
MSDU	MAC Service Data Unit.
OSAL	Operating System Abstraction Layer.
PAN	Personal Area Network.
PAN Coordinator	A coordinator that is the principal coordinator of a PAN.
PHY	Physical layer.
PIB	PAN Information Base.
RFD	Reduced Function Device.
RSSI	Received Signal Strength Indicator
SFD	Start of Frame Delimiter

2. API Overview

2.1 Interface Mechanisms

The following interface mechanisms are used in the MAC API.

2.1.1 Message Passing Function Calls

These API functions provide a message passing interface to the MAC by sending an OSAL message to the MAC event handler. Unless otherwise noted, the API functions described in this document are message passing functions. These functions do not contain critical sections and do not directly access MAC data.

2.1.2 Direct Execute Function Calls

These API functions directly execute code that performs a MAC operation. The function executes in the context of the caller. These functions may have critical sections and may directly access MAC data.

2.1.3 Callback Functions

These functions must be implemented by the application and are used to pass events and data from the MAC to the application. Data accessed through callback function parameters (such as a pointer to data) are only valid for the execution of the function and should not be considered valid when the function returns. These functions execute in the context of the MAC. The callback function implementation should avoid using critical sections and CPU intensive operations.

2.2 Zero Copy Data Interfaces

The interfaces for sending and receiving data between the MAC and the application are designed to not require a data copy, i.e. they are "zero copy". This results in a more CPU-efficient implementation. However the application must follow certain rules on when to allocate and deallocate data buffers, as described in Section 4.

2.3 Common Constants and Structures

2.3.1 Address Type

The following data types and values are used for the MAC address:

```
typedef uint8 sAddrExt_t[8];

typedef struct
{
  union
  {
    uint16      shortAddr;
    sAddrExt_t  extAddr;
  } addr;
  uint8         addrMode;
} sAddr_t;
```

addr.shortAddr – The short 16-bit MAC address.

addr.extAddr – The extended 64-bit MAC address.

addrMode – The address mode using the following values:

NAME	DESCRIPTION
SADDR_MODE_NONE	Address not present.
SADDR_MODE_SHORT	Short 16-bit address is used.
SADDR_MODE_EXT	Extended 64-bit address is used.

2.3.2 Status

The following status values are used in various API functions and callback function events.

2.3.2.1 Standard Status Values

NAME	DESCRIPTION
MAC_SUCCESS	Operation successful.
MAC_BEACON_LOSS	The beacon was lost following a synchronization request.
MAC_CHANNEL_ACCESS_FAILURE	The operation or data request failed because of activity on the channel.
MAC_COUNTER_ERROR	The frame counter purportedly applied by the originator of the received frame is invalid.
MAC_DENIED	The MAC was not able to enter low power mode.
MAC_DISABLE_TRX_FAILURE	Unused.
MAC_FRAME_TOO_LONG	The received frame or frame resulting from an operation or data request is too long to be processed by the MAC.
MAC_IMPROPER_KEY_TYPE	The key purportedly applied by the originator of the received frame is not allowed.
MAC_IMPROPER_SECURITY_LEVEL	The security level purportedly applied by the originator of the received frame does not meet the minimum security level.
MAC_INVALID_ADDRESS	The data request failed because neither the source address nor destination address parameters were present.
MAC_INVALID_GTS	Unused.
MAC_INVALID_HANDLE	The purge request contained an invalid handle.
MAC_INVALID_INDEX	Unused.
MAC_INVALID_PARAMETER	The API function parameter is out of range.
MAC_LIMIT_REACHED	The scan terminated because the PAN descriptor storage limit was reached.
MAC_NO_ACK	The operation or data request failed because no acknowledgement was received.
MAC_NO_BEACON	The scan request failed because no beacons were received or the orphan scan failed because no coordinator realignment was received.
MAC_NO_DATA	The associate request failed because no associate response was received or the poll request did not return any data.
MAC_NO_SHORT_ADDRESS	The short address parameter of the start request was invalid.
MAC_ON_TIME_TOO_LONG	Unused.
MAC_OUT_OF_CAP	Unused.
MAC_PAN_ID_CONFLICT	A PAN identifier conflict has been detected and communicated to the PAN coordinator.
MAC_PAST_TIME	Unused.
MAC_READ_ONLY	A set request was issued with a read-only identifier.
MAC_REALIGNMENT	A coordinator realignment command has been received.
MAC_SCAN_IN_PROGRESS	The scan request failed because a scan is already in progress.
MAC_SECURITY_ERROR	Cryptographic processing of the received secure frame failed.
MAC_SUPERFRAME_OVERLAP	The beacon start time overlapped the coordinator transmission time.
MAC_TRACKING_OFF	The start request failed because the device is not tracking the beacon of its coordinator.
MAC_TRANSACTION_EXPIRED	The associate response, disassociate request, or indirect data transmission failed because the peer device did not respond before the transaction expired or was purged.

MAC_TRANSACTION_OVERFLOW	The operation failed because MAC data buffers are full.
MAC_TX_ACTIVE	Unused.
MAC_UNAVAILABLE_KEY	The operation or data request failed because the security key is not available.
MAC_UNSUPPORTED_ATTRIBUTE	The set or get request failed because the attribute is not supported.
MAC_UNSUPPORTED_LEGACY	The received frame was secured with legacy security which is not supported.
MAC_UNSUPPORTED_SECURITY	The security of the received frame is not supported.

2.3.2.2 Proprietary Status Values

NAME	DESCRIPTION
MAC_UNSUPPORTED	The operation is not supported in the current configuration.
MAC_BAD_STATE	The operation could not be performed in the current state.
MAC_NO_RESOURCES	The operation could not be completed because no memory resources were available.

2.3.3 MAC Security Level

The following values indicate the MAC security level. The security level defines the encryption and/or authentication methods used on a message frame.

Note: Security is currently not implemented.

NAME	DESCRIPTION
MAC_SEC_LEVEL_NONE	No security is used.
MAC_SEC_LEVEL_MIC_32	MIC-32 authentication is used.
MAC_SEC_LEVEL_MIC_64	MIC-64 authentication is used.
MAC_SEC_LEVEL_MIC_128	MIC-128 authentication is used.
MAC_SEC_LEVEL_ENC	AES encryption is used.
MAC_SEC_LEVEL_ENC_MIC_32	AES encryption and MIC-32 authentication are used.
MAC_SEC_LEVEL_ENC_MIC_64	AES encryption and MIC-64 authentication are used.
MAC_SEC_LEVEL_ENC_MIC_128	AES encryption and MIC-128 authentication are used.

2.3.4 Key Identifier Mode

The following values indicate the key identifier mode. The key identifier mode defines how the key is determined from the key index.

NAME	DESCRIPTION
MAC_KEY_ID_MODE_NONE	Key is not used.
MAC_KEY_ID_MODE_IMPLICIT	Key is determined implicitly.
MAC_KEY_ID_MODE_1	Key is determined from the 1-byte key index.
MAC_KEY_ID_MODE_4	Key is determined from the 4-byte key index.
MAC_KEY_ID_MODE_8	Key is determined from the 8-byte key index.

2.3.5 Security Type

The following data type is used for MAC security.

Note: Security is currently not implemented.

```
typedef struct
{
  uint8   keySource[MAC_KEY_SOURCE_MAX_LEN];  /* Key source */
  uint8   securityLevel;                      /* Security level */
  uint8   keyIdMode;                          /* Key identifier mode */
  uint8   keyIndex;                           /* Key index */
} macSec_t;
```

keySource – This array identifies the originator of a key, as described in [1]. This parameter is ignored if `keyIdMode` is `MAC_KEY_ID_MODE_IMPLICIT` or `securityLevel` is `MAC_SEC_LEVEL_NONE`.

securityLevel – The security level. See 2.3.3. When set to `MAC_SEC_LEVEL_NONE` security is not used and all other parameters are ignored.

keyIdMode – The key ID mode. See 0. This parameter is ignored if `keyIdMode` is `MAC_KEY_ID_MODE_IMPLICIT` or `securityLevel` is `MAC_SEC_LEVEL_NONE`.

keyIndex – The key index, as described in [1]. This parameter is ignored if `keyIdMode` is `MAC_KEY_ID_MODE_IMPLICIT` or `securityLevel` is `MAC_SEC_LEVEL_NONE`.

2.3.6 MAC Timing Values

The 802.15.4 specification describes timing values based on the symbol period of the physical layer. This document adopts these same values as they relate to the MAC API. These values are shown in italic text in this document. These values are described below with their timing values converted to millisecond or microsecond units for the 2.4 GHz physical layer, which uses a 16 microsecond symbol period.

NAME	DESCRIPTION	SYMBOLS	2.4 GHz TIMING
aBaseSuperframeDuration	The number of symbols forming a superframe when the superframe order is zero.	960	15.36 ms
aUnitBackoffPeriod	The number of symbols forming the basic time period used by the CSMA-CA algorithm.	20	320 us

3. Initialization Interface

3.1 Introduction

The initialization interface is used to configure the MAC for support of certain optional features. Additional features are enabled by using the following API functions. All functions in the initialization interface are direct execute functions.

3.2 MAC_Init ( )

3.2.1 Description

This function initializes the MAC subsystem. It must be called once when the software system is started and before any other function in the MAC API is called.

3.2.2 Prototype

```
void MAC_Init(void)
```

3.2.3 Parameter Details

None.

3.2.4 Return

None.

3.3 MAC_InitDevice ( )

3.3.1 Description

This function initializes the MAC to associate with a non beacon-enabled network. This function would be used to initialize a device as an RFD. If this function is used it must be called during application initialization before any other function in the data or management API is called.

3.3.2 Prototype

```
void MAC_InitDevice(void)
```

3.3.3 Parameter Details

None.

3.3.4 Return

None.

3.4 MAC_InitCoord ( )

3.4.1 Description

This function initializes the MAC for operation as a coordinator. A coordinator can start a network, accept associate requests from other devices, send beacons, send indirect data, and other operations. This function would be used to initialize a device as an FFD. If this function is used it must be called during application initialization before any other function in the data or management API is called.

3.4.2 Prototype

```
void MAC_InitCoord(void)
```

3.4.3 Parameter Details

None.

3.4.4 Return

None.

3.5 MAC_InitSecurity ( )

This function initializes the MAC to allow use of security. If this function is used it must be called during application initialization before any other function in the data or management API is called.

Note: Security is currently not implemented.

3.5.1 Description

3.5.2 Prototype

```
void MAC_InitSecurity(void)
```

3.5.3 Parameter Details

None.

3.5.4 Return

None.

3.6 MAC_InitBeaconCoord ( )

3.6.1 Description

This function initializes the MAC for operation as a coordinator in a beacon-enabled network. If this function is used it must be called during application initialization before any other function in the data or management API is called.

3.6.2 Prototype

```
void MAC_InitBeaconCoord(void)
```

3.6.3 Parameter Details

None.

3.6.4 Return

None.

3.7 MAC_InitBeaconDevice ( )

3.7.1 Description

This function initializes the MAC to allow it to associate with and track a beacon-enabled network. If this function is used it must be called during application initialization before any other function in the data or management API is called.

3.7.2 Prototype

```
void MAC_InitBeaconDevice(void)
```

3.7.3 Parameter Details

None.

3.7.4 Return

None.

3.8 Example Configurations

Non beacon-enabled network or Zigbee configurations:

802.15.4 DEVICE FOR NON BEACON-ENABLED NETWORKS or ZIGBEE END DEVICE	802.15.4 COORDINATOR FOR NON BEACON-ENABLED NETWORKS or ZIGBEE ROUTER	802.15.4 PAN COORDINATOR FOR NON BEACON-ENABLED NETWORKS or ZIGBEE COORDINATOR
`MAC_InitDevice();`	`MAC_InitDevice();` `MAC_InitCoord();`	`MAC_InitCoord();`

802.15.4 Non beacon-enabled network configurations:

802.15.4 DEVICE FOR BEACON-ENABLED NETWORKS	802.15.4 COORDINATOR FOR BEACON-ENABLED NETWORKS	802.15.4 PAN COORDINATOR FOR BEACON-ENABLED NETWORKS

`MAC_InitDevice();`
`MAC_InitBeaconDevice();`
	`MAC_InitDevice();`
	`MAC_InitCoord();`
	`MAC_InitBeaconDevice();`
	`MAC_InitBeaconCoord();`
		`MAC_InitCoord();`
		`MAC_InitBeaconCoord();`

4. Data Interface

4.1 Introduction

This API provides interfaces to send and receive data between the MAC and the application.

4.2 Common Constants and Structures

4.2.1 Data header type

The following data type is used by the data interface to facilitate the zero copy mechanism.

```
typedef struct
{
  uint8     *p;
  uint8     len;
} sData_t;
```

p – Pointer to data.

len – Length of data in bytes.

4.2.2 Data Constants

The following constants are used to set up data buffers for `MAC_McpsDataReq()`.

NAME	VALUE	DESCRIPTION
MAC_MAX_FRAME_SIZE	102	Maximum application data length without security.
MAC_DATA_OFFSET	24	Data offset required for MAC header.
MAC_ENC_OFFSET	5	Data offset required for encryption header.
MAC_MIC_32_LEN	4	Length required for 32-bit authentication code.
MAC_MIC_64_LEN	8	Length required for 64-bit authentication code.
MAC_MIC_128_LEN	16	Length required for 128-bit authentication code.

4.3 Function Calls

4.3.1 MAC_McpsDataReq ( )

4.3.1.1 Description

This function sends application data to the MAC for transmission in a MAC data frame.

The MAC can only buffer a certain number of data request frames. When the MAC is congested and cannot accept the data request it sends a `MAC_MCPS_DATA_CNF` with status `MAC_TRANSACTION_OVERFLOW`. Eventually the MAC will become uncongested and send a `MAC_MCPS_DATA_CNF` for a buffered request. At this point the application can attempt another data request. Using this scheme, the application can send data whenever it wants but it must queue data to be resent if it receives an overflow status.

The data buffer passed to this function is allocated by the application. The application must not deallocate this buffer after calling this function; it must wait until it receives the `MAC_MCPS_DATA_CNF` for the frame.

The application sets data pointer `msdu.p` to point to a buffer containing the data the application is sending. Figure 1 shows how a buffer containing the data can be constructed.

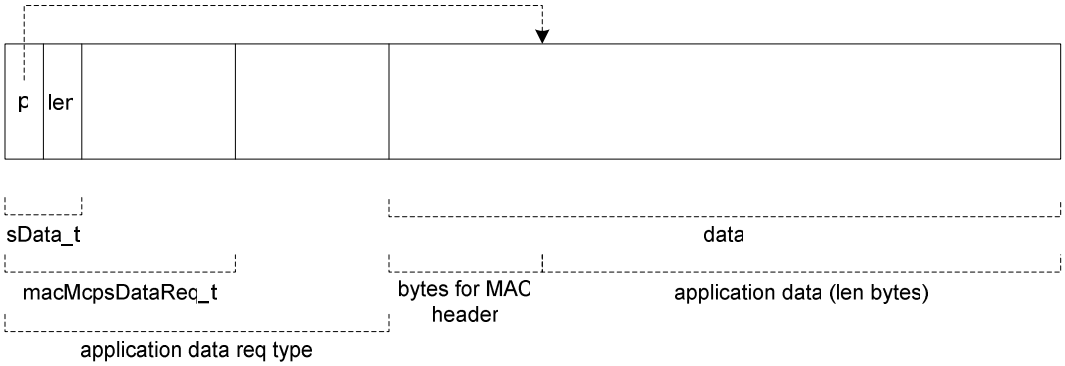

Figure 1.

As shown in the figure, the application must allocate a certain number of bytes before `msdu.p` to contain the MAC header. This number is `MAC_DATA_OFFSET` plus `MAC_ENC_OFFSET` if an encryption security suite is used.

Function `MAC_McpsDataAlloc()` can be used to simplify allocation and preparation of the data buffer. This function allocates a buffer of the correct size to contain the parameters, MAC header, and application data and prepares it as described in Figure 1.

The maximum data frame length is `MAC_MAX_FRAME_SIZE` bytes. This is reduced by `MAC_ENC_OFFSET` if encryption is used. If authentication is used, the data frame length is additionally reduced by the length of the authentication data, as shown in Section 4.2.2. For example, if AES-MIC-128 is used the maximum length is:

```
max = MAC_MAX_FRAME_SIZE - MAC_ENC_OFFSET - MAC_MIC_128_LEN;
```

4.3.1.2 Prototype

```
void MAC_McpsDataReq(macMcpsDataReq_t *pData)
```

4.3.1.3 Parameter Details

Parameters are passed using the following data structures.

```
typedef struct
{
  sAddr_t         dstAddr;
  uint16          dstPanId;
  uint8           srcAddrMode;
  uint8           mdsuHandle;
  uint8           txOptions;
  uint8           channel;
  uint8           power;
} macDataReq_t;


typedef struct
{
  macEventHdr_t   hdr;
  sData_t         msdu;
  macTxIntData_t  internal;
  macSec_t        sec;
  macDataReq_t    mac;
} macMcpsDataReq_t;
```


hdr – For internal use only.

msdu.p – Pointer to application data.

msdu.len – Length of data in bytes.

internal – For internal use only.

sec – Security parameters. See 2.3.5.

mac.dstAddr – The address of the destination device.

mac.dstPanId – The PAN ID of the destination device.

mac.srcAddrMode – The source address mode.

mac.mdsuHandle – Application-defined handle value associated with this data request.

mac.txOptions – TX options bit mask, a combination of the following values:

NAME	DESCRIPTION
MAC_TXOPTION_ACK	Acknowledged transmission. The MAC will attempt to retransmit the frame until it is acknowledged.
MAC_TXOPTION_GTS	GTS transmission (unused).
MAC_TXOPTION_INDIRECT	Indirect transmission. The MAC will queue the data and wait for the destination device to poll for it. This can only be used by a coordinator.
MAC_TXOPTION_NO_RETRANS	No frame retransmission. This proprietary option prevents the frame from being retransmitted.
MAC_TXOPTION_NO_CNF	No confirm. This proprietary option prevents a MAC_MCPS_DATA_CNF event from being sent for this frame.
MAC_TXOPTION_ALT_BE	Use PIB value MAC_ALT_BE for the minimum backoff exponent.
MAC_TXOPTION_PWR_CHAN	Use the power and channel values in macDataReq_t instead of the PIB values.

mac.channel – Transmit the data frame on this channel. This value is ignored if MAC_TXOPTION_PWR_CHAN is not used.

mac.power – Transmit the data frame at this power level. This value is ignored if MAC_TXOPTION_PWR_CHAN is not used.

4.3.1.4 Return

None.

4.3.2 MAC_McpsPurgeReq ( )

4.3.2.1 Description

This function purges and discards a data request from the MAC data queue. When the operation is complete the MAC sends a `MAC_MCPS_PURGE_CNF`.

4.3.2.2 Prototype

```
void MAC_McpsPurgeReq(uint8 msduHandle)
```

4.3.2.3 Parameter Details

msduHandle – The application-defined handle value associated with the data request.

4.3.2.4 Return

None.

4.3.3 MAC_McpsDataAlloc ( )

4.3.3.1 Description

This direct-execute function simplifies the allocation and preparation of the data buffer `MAC_McpsDataReq()`. The function allocates a buffer and prepares the data pointer as described in Section 4.3.1.

If the application allocates a buffer using this function and does use the buffer to call `MAC_McpsDataReq()` it must deallocate the buffer by calling `osal_msg_deallocate(pBuffer)`.

This function is a proprietary function and not defined in the 802.15.4 specification.

4.3.3.2 Prototype

```
macMcpsDataReq_t *MAC_McpsDataAlloc(uint8 len, uint8 securityLevel,
                                    uint8 keyIdMode)
```

4.3.3.3 Parameter Details

len – Length of application data in bytes.

securityLevel – Security level used for this frame. See Section 2.3.3. Set to `MAC_SEC_LEVEL_NONE` if security is not used.

keyIdMode – Key ID mode used for this frame. See Section 0. Set to `MAC_KEY_ID_MODE_NONE` if security is not used.

4.3.3.4 Return

Returns a pointer to the allocated buffer. If the function fails for any reason it returns NULL. The following parameters in the returned `macMcpsDataReq_t` structure are set by the function:

msdu.p – Points to the start of the application data area of the buffer.

msdu.len – Set to the `len` parameter passed to the function.

4.4 Callback Function Events

These events are sent to the application via the MAC callback function described in Section 0.

4.4.1 MAC_MCPS_DATA_IND

4.4.1.1 Description

This event sends data from the MAC to the application.

The parameters for this event point to a dynamically allocated memory buffer. When the application is finished using the data it must deallocate the buffer by calling `osal_msg_deallocate(pData)`.

When the MAC allocates a buffer for received data it can allocate extra space in the beginning of the buffer for application-defined data. The size of this application-defined data is set by calling `MAC_MlmeSetReq()` with attribute `MAC_DATA_IND_OFFSET`.

4.4.1.2 Parameter Details

Parameters are passed using the following data structures.

```
typedef struct
{
  sAddr_t        srcAddr;
  sAddr_t        dstAddr;
  uint32         timestamp;
  uint16         timestamp2;
  uint16         srcPanId;
  uint16         dstPanId;
  uint8          mpduLinkQuality;
  uint8          correlation;
  uint8          rssi;
  uint8          dsn;
} macDataInd_t;

typedef struct
{
  macEventHdr_t  hdr;
  sData_t        msdu;
  macRxIntData_t internal;
  macSec_t       sec;
  macDataInd_t   mac;
} macMcpsDataInd_t;
```

hdr – For internal use only.

msdu.p – Pointer to data received by the MAC.

msdu.len – Length of data in bytes.

internal – For internal use only.

sec – Security parameters. See 2.3.5.

mac.srcAddr – The address of the sending device.

mac.dstAddr – The address of the destination device.

mac.timestamp – The time, in *aUnitBackoffPeriod* units, at which the frame was received.

mac.timestamp2 – The time, in internal MAC timer units, at which the frame was received.

mac.srcPanId – The PAN ID of the sending device.

mac.dstPanId – The PAN ID of the destination device.

mac.mpduLinkQuality – The link quality of the received data frame. The value is based on the energy detect calculation, with values ranging linearly from 0x00 to 0xFF with the higher value indicating higher link quality.

mac.correlation – The raw correlation value of the received data frame. This value depends on the radio. See the chip data sheet for details.

mac.rssi – The received RF power in units of dBm.

dsn – The data sequence number of the received frame.

4.4.2 MAC_MCPS_DATA_CNF

4.4.2.1 Description

This event is sent to the application every time `MAC_McpsDataReq()` is called. The event returns the status of the data request. The event also passes back a pointer to the data buffer. The application can use this pointer to deallocate the buffer.

4.4.2.2 Parameter Details

Parameters are passed in the following structure:

```
typedef struct
{
  macEventHdr_t      hdr;
  uint8              msduHandle;
  macMcpsDataReq_t   *pDataReq;
  uint32             timestamp;
  uint16             timestamp2;
} macMcpsDataCnf_t;
```

hdr.status – The status of the data request operation, as follows:

NAME	DESCRIPTION
MAC_SUCCESS	Operation successful.
MAC_CHANNEL_ACCESS_FAILURE	Data transmission failed because of congestion on the channel.
MAC_FRAME_TOO_LONG	Data is too long to be processed by the MAC.
MAC_INVALID_PARAMETER	The API function parameter is out of range.
MAC_NO_ACK	No acknowledgement was received from the peer device.
MAC_TRANSACTION_EXPIRED	Indirect data transmission failed because the peer device did not respond before the transaction expired or was purged.
MAC_TRANSACTION_OVERFLOW	MAC data buffers are full.

msduHandle – Application-defined handle value associated with the data request.

pDataReq – Pointer to the data request buffer for this data confirm.

msduHandle – Application-defined handle value associated with the data request.

timestamp – The time, in *aUnitBackoffPeriod* units, at which the frame was transmitted.

timestamp2 – The time, in internal MAC timer units, at which the frame was transmitted.

4.4.3 MAC_MCPS_PURGE_CNF

4.4.3.1 Description

This event is sent to the application when `MAC_McpsPurgeReq()` is called. The event returns the status of the purge request.

4.4.3.2 Parameter Details

Parameters are passed in the following structure.

```
typedef struct
{
  macEventHdr_t      hdr;
  uint8              msduHandle;
} macMcpsPurgeCnf_t;
```

hdr.status – The status of the purge request, as follows:

NAME	DESCRIPTION
MAC_SUCCESS	Operation successful.
MAC_INVALID_HANDLE	The purge request contained an invalid handle.

pDataReq – Pointer to the data request buffer for this purge confirm.

msduHandle – Application-defined handle value associated with the data request.

5. Management Interface

5.1 Introduction

The following interfaces are used for 802.15.4 network management.

5.2 Common Constants and Structures

5.2.1 Channel Masks

The following mask values are used in combination to indicate a set of channels. For example, to use channels 11, 12, and 23 do:

```
uint32 chan = MAC_CHAN_11_MASK | MAC_CHAN_12_MASK | MAC_CHAN_23_MASK;
```

NAME	DESCRIPTION
MAC_CHAN_11_MASK – MAC_CHAN_28_MASK	Channel 11 through 28 mask.

Channels 27 and 28 are not defined by [1]. These channels may be outside the 2.4 GHz ISM band in some countries.

5.2.2 Channels

These values indicate a particular logical channel.

NAME	DESCRIPTION
MAC_CHAN_11 – MAC_CHAN_28	Channel 11 through 28.

Channels 27 and 28 are not defined by [1]. These channels may be outside the 2.4 GHz ISM band in some countries.

5.2.3 Channel Page

These values indicate the channel page.

Note: Only channel page 0 in the 2.4 GHz band is supported.

NAME	DESCRIPTION
MAC_CHANNEL_PAGE_0	2.4 GHz band using O-QPSK.
MAC_CHANNEL_PAGE_1	868 and 915 MHz bands using ASK.
MAC_CHANNEL_PAGE_2	868 and 915 MHz bands using O-QPSK.

5.2.4 Capability Information

This bit mask indicates the capability information of the device. This information is used during the association procedure.

NAME	DESCRIPTION
MAC_CAPABLE_PAN_COORD	Device is capable of becoming a PAN coordinator.
MAC_CAPABLE_FFD	Device is an FFD.
MAC_CAPABLE_MAINS_POWER	Device is mains powered rather than battery powered.
MAC_CAPABLE_RX_ON_IDLE	Device has its receiver on when idle.
MAC_CAPABLE_SECURITY	Device is capable of sending and receiving secured frames.
MAC_CAPABLE_ALLOC_ADDR	Request allocation of a short address in the associate procedure.

5.2.5 Get and Set Attributes

The following values are used to get and set MAC attributes using functions `MAC_MlmeGetReq()` and `MAC_MlmeSetReq()`. The default values for an attribute, if any, are shown in bold text. Read-only attributes are indicated by a ● symbol in the "R" column.

5.2.5.1 Standard Attributes

NAME	TYPE	RANGE	R	DESCRIPTION
MAC_ACK_WAIT_DURATION	uint8	**54**	●	The maximum number of symbols to wait for an acknowledgment frame.
MAC_ASSOCIATION_PERMIT	bool	TRUE, **FALSE**		TRUE if a coordinator is currently allowing association.
MAC_ASSOCIATED_PAN_COORD	bool	TRUE, **FALSE**		TRUE if the device is associated to the PAN coordinator.
MAC_AUTO_REQUEST	bool	**TRUE**, FALSE		TRUE if a device automatically sends a data request if its address is listed in the beacon frame
MAC_BATT_LIFE_EXT	bool	TRUE, **FALSE**		TRUE if battery life extension is enabled.
MAC_BATT_LIFE_EXT_PERIODS	uint8	**6**	●	The number of backoff periods during which the receiver is enabled following a beacon in battery life extension mode.
MAC_BEACON_PAYLOAD	uint8 *	pointer		The contents of the beacon payload.
MAC_BEACON_PAYLOAD_LENGTH	uint8	**0**-52		The length in bytes of the beacon payload
MAC_BEACON_ORDER	uint8	0-**15**		How often the coordinator transmits a beacon. See Section 5.3.10.3.
MAC_BEACON_TX_TIME	uint32	0x000000 – 0xffffff	●	The time the device transmitted its last beacon frame, in *aUnitBackoffPeriod* units.
MAC_BSN	uint8	0x00 – 0xff		The beacon sequence number.
MAC_COORD_EXTENDED_ADDRESS	sAddrExt_t	64-bit address		The extended address of the coordinator with which the device is associated.
MAC_COORD_SHORT_ADDRESS	uint16	0x0000 – **0xffff**		The short address assigned to the coordinator with which the device is

				associated. A value of MAC_ADDR_USE_EXT indicates that the coordinator is using its extended address.
MAC_DSN	uint8	0x00 – 0xff		The data or MAC command frame sequence number.
MAC_GTS_PERMIT	bool	TRUE, **FALSE**		TRUE if the PAN coordinator accepts GTS requests.
MAC_MAX_BE	uint8	3-8, **5**		The maximum value of the backoff exponent in the CSMA-CA algorithm.
MAC_MAX_CSMA_BACKOFFS	uint8	0-5, **4**		The maximum number of backoffs the CSMA-CA algorithm will attempt before declaring a channel failure.
MAC_MAX_FRAME_TOTAL_WAIT_TIME	uint16	143-25776, **1220**		The maximum number of CAP symbols in a beacon-enabled PAN, or symbols in a non beacon-enabled PAN, to wait for a frame intended as a response to a data request command frame.
MAC_MAX_FRAME_RETRIES	uint8	0-7, **3**		The maximum number of retries allowed after a transmission failure.
MAC_MIN_BE	uint8	0-8, **3**		The minimum value of the backoff exponent in the CSMA-CA algorithm.
MAC_PAN_ID	uint16	0x0000 – **0xffff**		The PAN identifier. If this value is 0xffff, the device is not associated.
MAC_PROMISCUOUS_MODE	bool	TRUE, **FALSE**		TRUE if the MAC is in promiscuous mode.
MAC_RESPONSE_WAIT_TIME	uint8	2-64, **32**		The maximum number of *aBaseSuperframeDuration* intervals a device shall wait for an associate response after transmitting an associate request.
MAC_RX_ON_WHEN_IDLE	bool	TRUE, **FALSE**		TRUE if the MAC enables its receiver during idle periods.
MAC_SECURITY_ENABLED	bool	TRUE, **FALSE**		TRUE if security is enabled.
MAC_SHORT_ADDRESS	uint16	0x0000 – **0xffff**		The short address that the device uses to communicate in the PAN. If the device is a PAN coordinator, this value shall be set before calling MAC_StartReq(). Otherwise the value is allocated during association. Value MAC_ADDR_USE_EXT indicates that the device is associated but not using a short address.
MAC_SUPERFRAME_ORDER	uint8	0-**15**		This specifies the length of the active portion of the superframe. See Section 5.3.10.3.
MAC_SYNC_SYMBOL_OFFSET	uint8	0	•	The timestamp offset from SFD in symbols.
MAC_TIMESTAMP_SUPPORTED	bool	**TRUE**, FALSE	•	TRUE if the MAC supports RX and TX timestamps.
MAC_TRANSACTION_PERSISTENCE_TIME	uint16	0x0000 – 0xffff, **0x01f4**		The maximum time in beacon intervals that a transaction is stored by a coordinator and indicated in the beacon.

5.2.5.2 Proprietary Attributes

NAME	TYPE	RANGE	R	DESCRIPTION
MAC_PHY_TRANSMIT_POWER	uint8	**0**-25		The transmit power in units of -1 dBm.
MAC_LOGICAL_CHANNEL	uint8	**11**-26		The logical channel.
MAC_EXTENDED_ADDRESS	sAddrExt_t	64-bit address		The extended address of the device.
MAC_ALT_BE	uint8	0-8, **1**		The alternate minimum value of the backoff exponent in the CSMA-CA algorithm.

5.2.6 Disassociate Reason

This value indicates whether the coordinator or the device itself is the reason for the disassociate procedure.

NAME	DESCRIPTION
MAC_DISASSOC_COORD	The coordinator wishes the device to disassociate.
MAC_DISASSOC_DEVICE	The device itself wishes to disassociate.

5.2.7 Scan Type

The following values indicate the type of scan.

NAME	DESCRIPTION
MAC_SCAN_ED	Energy detect scan. The device will tune to each channel and perform and energy measurement. The list of channels and their associated measurements will be returned at the end of the scan.
MAC_SCAN_ACTIVE	Active scan. The device tunes to each channel, sends a beacon request and listens for beacons. The PAN descriptors are returned at the end of the scan.
MAC_SCAN_PASSIVE	Passive scan. The device tunes to each channel and listens for beacons. The PAN descriptors are returned at the end of the scan.
MAC_SCAN_ORPHAN	Orphan scan. The device tunes to each channel and sends an orphan notification to try and find its coordinator. The status is returned at the end of the scan.

5.2.8 Special Address Values

The following special address values are used by the MAC.

NAME	DESCRIPTION
MAC_ADDR_USE_EXT	Short address value indicating extended address is used.
MAC_SHORT_ADDR_BROADCAST	Broadcast short address.
MAC_SHORT_ADDR_NONE	Short address when there is no short address.

5.2.9 Superframe Specification

The superframe specification is an unsigned 16-bit integer with the following contents:

Bits: 0-3	4-7	8-11	12	13	14	15
Beacon order	Superframe order	Final CAP slot	Battery life extension	Reserved	PAN coordinator	Association permit

The following macros are provided to help parse the superframe specification:

`MAC_SFS_BEACON_ORDER(s)` returns the beacon order.

`MAC_SFS_SUPERFRAME_ORDER(s)` returns the superframe order.

`MAC_SFS_FINAL_CAP_SLOT(s)` returns the final CAP slot.

`MAC_SFS_BLE(s)` returns the battery life extension bit.

`MAC_SFS_PAN_COORDINATOR(s)` returns the PAN coordinator bit.

`MAC_SFS_ASSOCIATION_PERMIT(s)` returns the association permit bit.

5.2.10 PAN Descriptor

The PAN descriptor contains information about a network as received in a beacon frame. The PAN descriptor uses the following structure.

```
typedef struct
{
  sAddr_t       coordAddress;
  uint16        coordPanId;
  uint16        superframeSpec;
  uint8         logicalChannel;
  uint8         channelPage;
  bool          gtsPermit;
  uint8         linkQuality;
  uint32        timestamp;
  bool          securityFailure;
  macSec_t      sec;
} macPanDesc_t;
```

coordAddress – The address of the coordinator sending the beacon.

coordPanId – The PAN ID of the network.

superframeSpec – The superframe specification of the network. See Section 5.2.10.

logicalChannel – The logical channel of the network.

channelPage – The current channel page occupied by the network.

gtsPermit – TRUE if coordinator accepts GTS requests.

linkQuality– The link quality of the received beacon. The value ranges from 0x00 to 0xFF with the higher value indicating higher link quality. The distribution of link quality values is specific to the hardware implementation. See the chip data sheet for details.

timestamp – The time at which the beacon was received, in *aUnitBackoffPeriod* units.

securityFailure – Set to TRUE if there was an error in the security processing.

sec – Security parameters. See 2.3.5.

5.2.11 Pending Address Specification

The pending address specification is an 8-bit integer with the following format:

Bits: 0-2	3	4-6	7
Number of short addresses pending	Reserved	Number of extended addresses pending	Reserved

The following macros are provided the parse the pending address specification:

`MAC_PEND_NUM_SHORT(pendAddrSpec)` returns the number of short addresses.

`MAC_PEND_NUM_EXT(pendAddrSpec)` returns the number of extended addresses.

`MAC_PEND_FIELDS_LEN(pendAddrSpec)` returns the length in bytes of the pending address fields in the beacon.

5.3 Function Calls

5.3.1 MAC_MlmeAssociateReq ( )

5.3.1.1 Description

This function sends an associate request to a coordinator device. The application shall attempt to associate only with a PAN that is currently allowing association, as indicated in the results of the scanning procedure. In a beacon-enabled PAN the beacon order must be set by using `MAC_MlmeSetReq()` before making the call to `MAC_MlmeAssociateReq()`.

When the associate request is complete the MAC sends a `MAC_MLME_ASSOCIATE_CNF` to the application.

5.3.1.2 Prototype

```
void MAC_MlmeAssociateReq(macMlmeAssociateReq_t *pData)
```

5.3.1.3 Parameter Details

Parameters are passed using the following structure:

```
typedef struct
{
  uint8         logicalChannel;
  uint8         channelPage;
  sAddr_t       coordAddress;
  uint16        coordPanId;
  uint8         capabilityInformation;
  macSec_t      sec;
} macMlmeAssociateReq_t;
```

logicalChannel – The channel on which to attempt association. See Section 5.2.2.

channelPage – The channel page on which to attempt association. See Section 5.2.3.

coordAddress – Address of the coordinator with which to associate. See Section 2.3.1.

coordPanId – The identifier of the PAN with which to associate.

capabilityInformation – The operational capabilities of this device. See Section 5.2.3.

sec – The security parameters for this message. See 2.3.5.

5.3.1.4 Return

None.

5.3.2 MAC_MlmeAssociateRsp ( )

5.3.2.1 Description

This function sends an associate response to a device requesting to associate. This function must be called after receiving a `MAC_MLME_ASSOCIATE_IND`. When the associate response is complete the MAC sends a `MAC_MLME_COMM_STATUS_IND` to the application to indicate the success or failure of the operation.

5.3.2.2 Prototype

```
void MAC_MlmeAssociateRsp(macMlmeAssociateRsp_t *pData)
```

5.3.2.3 Parameter Details

Parameters are passed using the following structure:


```
typedef struct
{
  sAddrExt_t        deviceAddress;
  uint16            assocShortAddress;
  uint8             status;
  macSec_t          sec;
} macMlmeAssociateRsp_t;
```

deviceAddress – The address of the device requesting association.

assocShortAddress – The short address allocated to the device. This value is only set if association is successful and the device requested a short address.

status – The status of the association attempt. See Section 2.3.2.

sec – Security parameters. See 2.3.5.

5.3.2.4 Return

None.

5.3.3 MAC_MlmeDisassociateReq ( )

5.3.3.1 Description

This function is used by an associated device to notify the coordinator of its intent to leave the PAN. It is also used by the coordinator to instruct an associated device to leave the PAN. When the disassociate procedure is complete the MAC sends a `MAC_MLME_DISASSOCIATE_CNF` to the application.

5.3.3.2 Prototype

```
void MAC_MlmeDisassociateReq(macMlmeDisassociateReq_t *pData)
```

5.3.3.3 Parameter Details

Parameters are passed using the following structure:

```
typedef struct
{
  sAddr_t           deviceAddress;
  uint16            devicePanId;
  uint8             disassociateReason;
  bool              txIndirect;
  macSec_t          sec;
} macMlmeDisassociateReq_t;
```

deviceAddress – The address of the device with which to disassociate.

devicePanId – The PAN ID of the device.

disassociateReason – The disassociate reason. See Section 5.2.6.

txIndirect – Set to TRUE if the disassociation notification command is to be sent indirectly.

sec – The security parameters for this message. See 2.3.5.

5.3.3.4 Return

None.

5.3.4 MAC_MlmeGetReq ( )

5.3.4.1 Description

This direct execute function retrieves an attribute value from the MAC PIB.

5.3.4.2 Prototype

```
uint8 MAC_MlmeGetReq(uint8 pibAttribute, void *pValue)
```

5.3.4.3 Parameter Details

pibAttribute – The attribute identifier. See Section 5.2.5.

pValue – pointer to the attribute value. The data type for each attribute value is shown in Section 5.2.5.

5.3.4.4 Return

The status of the request, as follows:

NAME	DESCRIPTION
MAC_SUCCESS	Operation successful.
MAC_UNSUPPORTED_ATTRIBUTE	Attribute not found.

5.3.5 MAC_MlmeOrphanRsp ( )

5.3.5.1 Description

This function is called in response to an orphan notification from a peer device. This function must be called after receiving a MAC_MLME_ORPHAN_IND. When the orphan response is complete the MAC sends a MAC_MLME_COMM_STATUS_IND to the application to indicate the success or failure of the operation.

5.3.5.2 Prototype

```
void MAC_MlmeOrphanRsp(macMlmeOrphanRsp_t *pData)
```

5.3.5.3 Parameter Details

Parameters are passed using the following structure:

```
typedef struct
{
  sAddrExt_t        orphanAddress;
  uint16            shortAddress;
  bool              associatedMember;
  macSec_t          sec;
} macMlmeOrphanRsp_t;
```

orphanAddress – The extended address of the device sending the orphan notification.

shortAddress – The short address of the orphaned device. This value is only used if associatedMember is TRUE. The value MAC_ADDR_USE_EXT indicates the extended address is used instead of the short address.

associatedMember – Set to TRUE if the orphaned device is associated with this coordinator.

sec – The security parameters for this message. See 2.3.5.

5.3.5.4 Return

None.

5.3.6 MAC_MlmePollReq ( )

5.3.6.1 Description

This function is used to request pending data from the coordinator. When the poll request is complete the MAC sends a MAC_MLME_POLL_CNF to the application. If a data frame of nonzero length is received from the coordinator the MAC sends a MAC_MLME_POLL_CNF with status MAC_SUCCESS and then sends a MAC_MCPS_DATA_IND with the data.

5.3.6.2 Prototype

```
void MAC_MlmePollReq(macMlmePollReq_t *pData)
```

5.3.6.3 Parameter Details

Parameters are passed using the following structure:

```
typedef struct
{
  sAddr_t          coordAddress;
  uint16           coordPanId;
  macSec_t         sec;
} macMlmePollReq_t;
```

coordAddress – The address of the coordinator device to poll.

coordPanId – The PAN ID of the coordinator.

sec – The security parameters for this message. See 2.3.5.

5.3.6.4 Return

None.

5.3.7 MAC_MlmeResetReq ( )

This direct execute function resets the MAC. This function must be called once at system startup with setDefaultPib set to TRUE before any other function in the management API is called.

5.3.7.1 Description

5.3.7.2 Prototype

```
uint8 MAC_MlmeResetReq(bool setDefaultPib)
```

5.3.7.3 Parameter Details

setDefaultPib – Set to TRUE to reset the MAC PIB to its default values.

5.3.7.4 Return

Returns MAC_SUCCESS always.

5.3.8 MAC_MlmeScanReq ( )

5.3.8.1 Description

This function initiates an energy detect, active, passive, or orphan scan on one or more channels. An energy detect scan measures the peak energy on each requested channel. An active scan sends a beacon request on each channel and then listening for beacons. A passive scan is a receive-only operation that listens for beacons on each channel. An orphan scan is used to locate the coordinator with which the scanning device had previously associated. When a scan operation is complete the MAC sends a `MAC_MLME_SCAN_CNF` to the application.

For active or passive scans the application sets the `maxResults` parameter the maximum number of PAN descriptors to return. If `maxResults` is greater than zero then the application must also set `result.panDescriptor` to point to a buffer of size `maxResults * sizeof(macPanDesc_t)` to store the results of the scan. The application must not access or deallocate this buffer until it receives a `MAC_MLME_SCAN_CNF`. The MAC will store up to `maxResults` PAN descriptors and ignore duplicate beacons.

An alternative way to get results for an active or passive scan is to set `maxResults` to zero or set PIB attribute `MAC_AUTO_REQUEST` to FALSE. Then the MAC will not store results but rather send a `MAC_MLME_BEACON_NOTIFY_IND` for each beacon received. The application will not need to supply any memory to store the scan results but the MAC will not filter out duplicate beacons.

For energy detect scans the application must set `result.energyDetect` to point to a buffer of size 18 bytes to store the results of the scan. The application must not access or deallocate this buffer until is receives a `MAC_MLME_SCAN_CNF`.

An energy detect, active or passive scan may be performed at any time if a scan is not already in progress. However a device cannot perform any other MAC management operation or send or receive MAC data until the scan is complete.

5.3.8.2 Prototype

```
void MAC_MlmeScanReq(macMlmeScanReq_t *pData)
```

5.3.8.3 Parameter Details

Parameters are passed using the following structure:

```
typedef struct
{
  uint32           scanChannels;
  uint8            scanType;
  uint8            scanDuration;
  uint8            channelPage;
  uint8            maxResults;
  macSec_t         sec;
  {
    uint8         *pEnergyDetect;
    macPanDesc_t  *pPanDescriptor;
  } result;
} macMlmeScanReq_t;
```

scanChannels – Bit mask indicating which channels to scan. See Section 5.2.1.

scanType – The type of scan. See Section 5.2.7.

scanDuration – The exponent used in the scan duration calculation. The scan duration is calculated as follows:

scan duration (ms) = (*aBaseSuperframeDuration* ms) * ($2^{\text{scanDuration}} + 1$)

Valid range is 0-14.

channelPage – The channel page on which to perform the scan. See Section 5.2.3.

maxResults – The maximum number of PAN descriptor results to return for an active or passive scan. This parameter is ignored for energy detect and orphan scans.

sec – The security parameters for orphan scan. See 2.3.5.

result.pEnergyDetect – Pointer to a buffer to store energy detect measurements. This value is only used for energy detect scans.

result.pPanDescriptor – Pointer to a buffer to store PAN descriptors. This value is only used for active and passive scans.

5.3.8.4 Return

None.

5.3.9 MAC_MlmeSetReq ( )

This direct execute function sets an attribute value in the MAC PIB.

5.3.9.1 Description

5.3.9.2 Prototype

```
uint8 MAC_MlmeSetReq(uint8 pibAttribute, void *pValue)
```

5.3.9.3 Parameter Details

pibAttribute – The attribute identifier. See Section 5.2.5.

pValue – pointer to the attribute value. The data type for each attribute value is shown in Section 5.2.5.

5.3.9.4 Return

The status of the request, as follows:

NAME	DESCRIPTION
MAC_SUCCESS	Operation successful.
MAC_UNSUPPORTED_ATTRIBUTE	Attribute identifier unsupported.
MAC_INVALID_PARAMETER	Value out of range.
MAC_READ_ONLY	Attribute is read-only.

5.3.10 MAC_MlmeStartReq ( )

5.3.10.1 Description

This function is called by a coordinator or PAN coordinator to start or reconfigure a network. Before starting a network the device must have set its short address. A PAN coordinator sets the short address by setting the attribute `MAC_SHORT_ADDRESS`. A coordinator sets the short address through association.

When parameter `panCoordinator` is TRUE, the MAC automatically sets attributes `MAC_PAN_ID` and `MAC_LOGICAL_CHANNEL` to the `panId` and `logicalChannel` parameters. If `panCoordinator` is FALSE, these parameters are ignored (they would already be set through association).

The parameter `beaconOrder` controls whether the network is beacon-enabled or non beacon-enabled. For a beacon-enabled network this parameter also controls the beacon transmission interval.

When the operation is complete the MAC sends a `MAC_MLME_START_CNF` to the application.

5.3.10.2 Prototype

```
void MAC_MlmeStartReq(macMlmeStartReq_t *pData)
```

5.3.10.3 Parameter Details

Parameters are passed using the following structure:

```
typedef struct
{
  uint32           startTime;
  uint16           panId;
  uint8            logicalChannel;
  uint8            channelPage;
  uint8            beaconOrder;
  uint8            superframeOrder;
  bool             panCoordinator;
  bool             batteryLifeExt;
  bool             coordRealignment;
  macSec_t         realignSec;
  macSec_t         beaconSec;
} macMlmeStartReq_t;
```

startTime – The time to begin transmitting beacons relative to the received beacon. This parameter is ignored if the device is a PAN coordinator or when starting a non beacon-enabled network. The time is specified in symbol periods and is rounded to the nearest *aUnitBackoffPeriod* symbol periods.

panId – The PAN ID to use. This parameter is ignored if `panCoordinator` is FALSE.

logicalChannel – The logical channel to use. See Section 5.2.2. This parameter is ignored if `panCoordinator` is FALSE.

channelPage – The channel page to use. See Section 5.2.3. This parameter is ignored if `panCoordinator` is FALSE.

beaconOrder – The exponent used to calculate the beacon interval. The beacon interval is calculated as follows:

interval (ms) = (*aBaseSuperframeDuration* ms) * $2^{\text{beaconOrder}}$

Valid range is 0-14. For a non beacon-enabled network set to 15.

superframeOrder – The exponent used to calculate the superframe duration. The superfame duration is calculated with the same equation as the beacon interval. The `superframeOrder` must be less than or equal to the

`beaconOrder`. It can also be set to 15 to configure a network that sends a beacon but has no CAP. For a non beacon-enabled network this value is ignored.

panCoordinator – Set to TRUE to start a network as PAN coordinator.

batteryLifeExt – If this value is TRUE, the receiver is disabled after `MAC_BATT_LIFE_EXT_PERIODS` full backoff periods following the interframe spacing period of the beacon frame. This parameter is ignored for non beacon-enabled networks.

coordRealignment – Set to TRUE to transmit a coordinator realignment prior to changing the superframe configuration.

realignSec – Security parameters for the coordinator realignment frame. See 2.3.5. This parameter is ignored if `coordRealignment` is FALSE.

beaconSec – Security parameters for the beacon frame. See 2.3.5.

5.3.10.4 Return

None.

5.3.11 MAC_MlmeSyncReq ( )

5.3.11.1 Description

This function requests the MAC to synchronize with the coordinator by acquiring and optionally tracking its beacons. Synchronizing with the coordinator is recommended before associating in a beacon-enabled network. If the beacon could not be located on its initial search or during tracking, the MAC sends a MAC_MLME_SYNC_LOSS_IND to the application with status MAC_BEACON_LOSS.

Before calling this function the application must set PIB attributes MAC_BEACON_ORDER, MAC_PAN_ID and either MAC_COORD_SHORT_ADDRESS or MAC_COORD_EXTENDED_ADDRESS to the address of the coordinator with which to synchronize.

The application may wish to set PIB attribute MAC_AUTO_REQUEST to FALSE before calling this function. Then when the MAC successfully synchronizes with the coordinator it will send the application a MAC_MLME_BEACON_NOTIFY_IND. After receiving the event the application may set MAC_AUTO_REQUEST to TRUE to stop receiving beacon notifications.

This function is only applicable to beacon-enabled networks.

5.3.11.2 Prototype

```
void MAC_MlmeSyncReq(macMlmeSyncReq_t *pData)
```

5.3.11.3 Parameter Details

Parameters are passed using the following structure:

```
typedef struct
{
  uint8            logicalChannel;
  uint8            channelPage;
  bool             trackBeacon;
} macMlmeSyncReq_t;
```

logicalChannel – The logical channel to use. See Section 5.2.2.

channelPage – The channel page to use. See Section 5.2.3.

trackBeacon – Set to TRUE to continue tracking beacons after synchronizing with the first beacon. Set to FALSE to only synchronize with the first beacon. If already tracking beacons, set to FALSE to stop tracking.

5.3.11.4 Return

None.

5.4 Callback Function Events

5.4.1 MAC_MLME_ASSOCIATE_IND

5.4.1.1 Description

This event is sent to the application when the MAC receives an associate request from another device. The application must call `MAC_MlmeAssociateRsp()` after receiving this event. This event will only be sent to FFD applications which set PIB attribute `MAC_ASSOCIATION_PERMIT` to TRUE.

5.4.1.2 Parameter Details

Parameters are passed in the following structure:

```
typedef struct
{
  macEventHdr_t    hdr;
  sAddrExt_t       deviceAddress;
  uint8            capabilityInformation;
  macSec_t         sec;
} macMlmeAssociateInd_t;
```

hdr – The event header.

deviceAddress – The address of the device requesting association.

capabilityInformation – The operational capabilities of the device requesting association. See Section 5.2.3.

sec – The security parameters for this message. See 2.3.5.

5.4.2 MAC_MLME_ASSOCIATE_CNF

5.4.2.1 Description

This event is sent to the application in response to a `MAC_MlmeAssociateReq()`. The event indicates the status of the associate attempt. If the associate was successful and a short address was requested then the short address is included in the event. Otherwise the short address parameter is not valid.

5.4.2.2 Parameter Details

Parameters are passed in the following structure:

```
typedef struct
{
  macEventHdr_t    hdr;
  uint16           assocShortAddress;
  macSec_t         sec;
} macMlmeAssociateCnf_t;
```

hdr.status – The status of the associate attempt. Set to `MAC_SUCCESS` if successful. Otherwise, the status parameter indicates an error code received from the peer device (see Section 2.3.2) or one of the following error codes generated internally by the MAC:

NAME	DESCRIPTION
MAC_CHANNEL_ACCESS_FAILURE	Data transmission failed because of congestion on the channel.
MAC_INVALID_PARAMETER	The API function parameter is out of range.
MAC_NO_ACK	No acknowledgement was received from the peer device.
MAC_NO_DATA	No associate response was received from the peer device.

assocShortAddress – If successful, the short address allocated to this device.

sec – The security parameters for this message. See 2.3.5.

5.4.3 MAC_MLME_DISASSOCIATE_IND

5.4.3.1 Description

This event is sent to the application to indicate that the device has been disassociated from the network.

5.4.3.2 Parameter Details

Parameters are passed using the following structure:

```
typedef struct
{
  macEventHdr_t    hdr;
  sAddrExt_t       deviceAddress;
  uint8            disassociateReason;
  macSec_t         sec;
} macMlmeDisassociateInd_t;
```

deviceAddress – The address of the device sending the disassociate command.

disassociateReason – The disassociate reason. See Section 5.2.6.

sec – The security parameters for this message. See 2.3.5.

5.4.4 MAC_MLME_DISASSOCIATE_CNF

5.4.4.1 Description

This event is sent to the application in response to a `MAC_MlmeDisassociateReq()`. The event indicates the status of the disassociate attempt.

5.4.4.2 Parameter Details

Parameters are passed using the following structure:

```
typedef struct
{
  macEventHdr_t    hdr;
  sAddr_t          deviceAddress;
  uint16           panId;
} macMlmeDisassociateCnf_t;
```

hdr.status – The status of the disassociate attempt, as follows:

NAME	DESCRIPTION
MAC_SUCCESS	Operation successful.
MAC_CHANNEL_ACCESS_FAILURE	Data transmission failed because of congestion on the channel.
MAC_INVALID_PARAMETER	The API function parameter is out of range.
MAC_NO_ACK	No acknowledgement was received from the peer device.
MAC_TRANSACTION_EXPIRED	Transmission failed because the peer device did not respond before the transaction expired.
MAC_TRANSACTION_OVERFLOW	MAC data buffers are full.

deviceAddress – The address of the device that has either requested disassociation or been instructed to disassociate by its coordinator.

panId – The PAN ID of the device that has either requested disassociation or been instructed to disassociate by its coordinator.

5.4.5 MAC_MLME_BEACON_NOTIFY_IND

5.4.5.1 Description

This event is sent to the application when the MAC receives a beacon frame and the beacon contains payload data or attribute `MAC_AUTO_REQUEST` is set to FALSE. The event also contains an LQI measurement and the time the beacon was received in addition to the beacon information.

5.4.5.2 Parameter Details

Parameters are passed in the following structure:

```
typedef struct
{
  macEventHdr_t  hdr;
  uint8          bsn;
  macPanDesc_t   *pPanDesc;
  uint8          pendAddrSpec;
  uint8          *pAddrList;
  uint8          sduLength;
  uint8          *pSdu;
} macMlmeBeaconNotifyInd_t;
```

bsn – The beacon sequence number.

pPanDesc – The PAN descriptor for the received beacon. See Section 5.2.10.

pendAddrSpec – The beacon pending address specification. See Section 5.2.11.

pAddrList – The list of device addresses for which the sender of the beacon has data. The length of the list can be determined with macro `MAC_PEND_FIELDS_LEN()`, see Section 5.2.11.

sduLength – The number of bytes in the beacon payload of the beacon frame.

pSdu – The beacon payload.

5.4.6 MAC_MLME_ORPHAN_IND

5.4.6.1 Description

This event is sent to the application when the MAC receives an orphan notification from another device. The application must call `MAC_MlmeOrphanRsp()` after receiving this event. This event will only be sent to FFD applications.

5.4.6.2 Parameter Details

Parameters are passed using the following structure:

```
typedef struct
{
  macEventHdr_t   hdr;
  sAddrExt_t      orphanAddress;
  macSec_t        sec;
} macMlmeOrphanInd_t;
```

orphanAddress – The address of the orphaned device.

sec – Security parameters for this message. See 2.3.5.

5.4.7 MAC_MLME_SCAN_CNF

5.4.7.1 Description

This event is sent to the application in response to a `MAC_MlmeScanReq()` when the scan operation is complete. The event indicates the status of the scan. For an energy detect scan a list of energy measurements is returned. For an active or passive scan a list of PAN descriptors is returned.

5.4.7.2 Parameter Details

Parameters are passed using the following structure:

```
typedef struct
{
  macEventHdr_t   hdr;
  uint8           scanType;
  uint8           channelPage;
  uint32          unscannedChannels;
  uint8           resultListSize;
  union
  {
    uint8         *pEnergyDetect;
    macPanDesc_t  *pPanDescriptor;
  } result;
} macMlmeScanCnf_t;
```

hdr.status – The status of the scan request, as follows:

NAME	DESCRIPTION
MAC_SUCCESS	Operation successful.
MAC_INVALID_PARAMETER	The API function parameter is out of range.
MAC_NO_BEACON	The active or passive scan failed because no beacons were received or the orphan scan failed because no coordinator realignment was received.

scanType – The type of scan requested. See Section 5.2.7.

channelPage – The channel page of the scan. See Section 5.2.3.

unscannedChannels – Bit mask of channels that were not scanned. See Section 5.2.2.

resultListSize – The number of elements returned in the results list. This value is not used if `scanType` is `MAC_SCAN_ORPHAN`.

result.pEnergyDetect – The list of energy measurements, one for each channel scanned. This value is only used if scanType is `MAC_SCAN_ED`. The distribution of energy values is specific to the hardware implementation. See the hardware data sheet for details.

result.pPanDescriptor – The list of PAN descriptors, one for each beacon found. See Section 5.2.10. This value is only used if the `scanType` is `MAC_SCAN_ACTIVE` or `MAC_SCAN_PASSIVE`.

5.4.8 MAC_MLME_START_CNF

5.4.8.1 Description

This event is sent to the application in response to a `MAC_MlmeStartReq()`. The event indicates the status of the start request.

5.4.8.2 Parameter Details

Parameters are passed using the following structure:

```
typedef struct
{
  macEventHdr_t   hdr;
} macMlmeStartCnf_t;
```

hdr.status – The status of start request, as follows:

NAME	DESCRIPTION
MAC_SUCCESS	Operation successful.
MAC_INVALID_PARAMETER	The API function parameter is out of range.
MAC_NO_SHORT_ADDRESS	The device does not have a short address.
MAC_SUPERFRAME_OVERLAP	The beacon start time overlapped the coordinator transmission time.

5.4.9 MAC_MLME_SYNC_LOSS_IND

5.4.9.1 Description

This event is sent to the application when the MAC loses synchronization with the coordinator or has a PAN ID conflict. The status indicates the reason for the event.

5.4.9.2 Parameter Details

Parameters are passed using the following structure:

```
typedef struct
{
  macEventHdr_t   hdr;
  uint16          panId;
  uint8           logicalChannel;
  uint8           channelPage;
  macSec_t        sec;
} macMlmeSyncLossInd_t;
```

hdr.status – The reason that synchronization was lost, as follows:

NAME	DESCRIPTION
MAC_BEACON_LOSS	The beacon was lost following a synchronization request.
MAC_PAN_ID_CONFLICT	A PAN identifier conflict has been detected.
MAC_REALIGNMENT	A coordinator realignment command has been received.

panId – The PAN ID of the realignment.

logicalChannel – The logical channel of the realignment.

channelPage – The channel page of the realignment.

sec – Security parameters for this message. See 2.3.5.

5.4.10 MAC_MLME_POLL_CNF

5.4.10.1 Description

This event is sent to the application in response to a `MAC_MlmePollReq()`. If the poll request was successful and data was received the status is set to `MAC_SUCESS`. If the poll request was successful and no data was received the status is set to `MAC_NO_DATA`. Other status values indicate failure as described below.

5.4.10.2 Parameter Details

Parameters are passed using the following structure:

```
typedef struct
{
  macEventHdr_t   hdr;
} macMlmePollCnf_t;
```

hdr.status – The status of the poll request, as follows:

NAME	DESCRIPTION
MAC_SUCCESS	Operation successful.
MAC_CHANNEL_ACCESS_FAILURE	Failed because of congestion on the channel.
MAC_INVALID_PARAMETER	The API function parameter is out of range.
MAC_NO_ACK	No acknowledgement was received from the peer device.
MAC_NO_DATA	No data was received from the peer device.

5.4.11 MAC_MLME_COMM_STATUS_IND

5.4.11.1 Description

This event is sent to the application for various reasons. It indicates the status of a `MAC_MlmeAssociateRsp()` or `MAC_MlmeOrphanRsp()`. It also indicates the MAC has received a secure frame that generated an error during security processing.

5.4.11.2 Parameter Details

Parameters are passed using the following structure:

```
typedef struct
{
  macEventHdr_t   hdr;
  sAddr_t         srcAddr;
  sAddr_t         dstAddr;
  uint16          panId;
  uint8           reason;
  macSec_t        sec;
} macMlmeCommStatusInd_t;
```

hdr.status – The status for this event, as follows:

NAME	DESCRIPTION
MAC_CHANNEL_ACCESS_FAILURE	The response frame failed because of activity on the channel.
MAC_FRAME_TOO_LONG	The response frame or received frame is too long to be processed by the MAC.
MAC_INVALID_PARAMETER	The API function parameter is out of range.
MAC_NO_ACK	The response frame failed because no acknowledgement was received.

MAC_TRANSACTION_EXPIRED	The response frame failed because the peer device did not respond before the transaction expired or was purged.
MAC_TRANSACTION_OVERFLOW	The response frame failed because MAC data buffers are full.

srcAddr – The source address associated with the event.

dstAddr – The destination address associated with the event.

panId – The PAN ID associated with the event.

reason – The reason the event was generated. This parameter is not defined in [1] but may be used to distinguish between the different uses of the event. Values are as follows:

NAME	DESCRIPTION
MAC_COMM_ASSOCIATE_RSP	Event sent in response to MAC_AssociateRsp().
MAC_COMM_ORPHAN_RSP	Event sent in response to MAC_OrphanRsp().
MAC_COMM_RX_SECURE	Event sent as a result of receiving a secure frame.

sec – Security parameters for this message. See 2.3.5.

5.4.12 MAC_MLME_POLL_IND

5.4.12.1 Description

This event is sent to the application only when the MAC is configured for queueing of indirect data in the application by setting macCfg.appPendingQueue to TRUE. If set to FALSE (the default), this event is never sent to the application.

This event is sent to the application when a data request command frame is received from another device. The application should call MAC_McpsDataReq() to send data to that device with TX option MAC_TXOPTION_NO_RETRANS set. If the application has no data for that device it should send zero-length data with TX options MAC_TXOPTION_NO_CNF and MAC_TXOPTION_NO_RETRANS set.

5.4.12.2 Parameter Details

Parameters are passed using the following structure:

```
typedef struct
{
  macEventHdr_t   hdr;
  uint16          srcShortAddr;
  uint16          srcPanId;
} macMlmePollInd_t;
```

srcShortAddr – The short address of the device requesting data.

srcPanId – The PAN ID of the device requesting data.

6. Extension Interface

6.1 Introduction

The extension interface provides an API for additional software features that are not defined by the 802.15.4 specification, such as power management.

6.2 Common Constants and Structures

6.2.1 Power Mode

The following values indicate the power mode of the MAC. The difference between "lite" and "deep" sleep modes is dependent on the hardware being used. In general, the wakeup time for deep sleep is longer than lite sleep. See the chip data sheet for details.

NAME	DESCRIPTION
MAC_PWR_ON	MAC and radio hardware is powered on.
MAC_PWR_SLEEP_LITE	MAC and radio hardware are partially powered off.
MAC_PWR_SLEEP_DEEP	MAC and radio hardware are fully powered off.

6.3 Function Calls

6.3.1 MAC_PwrOffReq ( )

6.3.1.1 Description

This direct execute function requests the MAC to power off the radio hardware and go to sleep. If the MAC is able to power off it will execute its power off procedure and return MAC_SUCCESS. If the MAC is unable to sleep it will return MAC_DENIED. The MAC is unable to sleep when it is executing certain procedures, such as a scan, data request, or association. If this function is called when the MAC is already in sleep mode it will return MAC_SUCCESS but do nothing.

When the MAC is in sleep mode the application must not call any MAC API function besides power management functions until the MAC is awake.

6.3.1.2 Prototype

```
uint8 MAC_PwrOffReq(uint8 mode)
```

6.3.1.3 Parameter Details

mode – The desired low power mode, either MAC_PWR_SLEEP_LITE or MAC_PWR_SLEEP_DEEP. See Section 6.2.1.

6.3.1.4 Return

The status of the request, as follows:

NAME	DESCRIPTION
MAC_SUCCESS	Operation successful; the MAC is powered off.
MAC_DENIED	The MAC was not able to power off.

6.3.2 MAC_PwrOnReq ( )

6.3.2.1 Description

This direct execute function requests the MAC to power on the radio hardware and wake up. The function powers on the radio hardware, blocking until completion.

6.3.2.2 Prototype

```
void MAC_PwrOnReq(void)
```

6.3.2.3 Parameter Details

None.

6.3.2.4 Return

None.

6.3.3 MAC_PwrMode ( )

6.3.3.1 Description

This function returns the current power mode of the MAC.

6.3.3.2 Prototype

```
uint8 MAC_PwrMode(void)
```

6.3.3.3 Parameter Details

None.

6.3.3.4 Return

The current power mode of the MAC. See Section 6.2.1.

6.3.4 MAC_PwrNextTimeout ( )

6.3.4.1 Description

This function returns the next MAC timer expiration in *aUnitBackoffPeriod* units. If no timer is running it returns zero.

6.3.4.2 Prototype

```
uint32 MAC_PwrNextTimeout(void)
```

6.3.4.3 Parameter Details

None.

6.3.4.4 Return

The next MAC timer expiration in *aUnitBackoffPeriod* units or zero if no timer is running.

6.3.5 MAC_RandomByte ( )

6.3.5.1 Description

This function returns a random byte from the MAC random number generator.

6.3.5.2 Prototype

```
uint8 MAC_RandomByte(void)
```

6.3.5.3 Parameter Details

None.

6.3.5.4 Return

A random byte.

6.4 Callback Function Events

6.4.1 MAC_PWR_ON_CNF

6.4.1.1 Description

This event is currently unused.

7. Callback Interface

7.1.1 MAC_CbackEvent ( )

7.1.1.1 Description

This callback function sends MAC events to the application.

The application must implement this function. A typical implementation of this function would allocate an OSAL message, copy the event parameters to the message, and send the message to the application's OSAL event handler. This function may be executed from task or interrupt context and therefore must be reentrant.

7.1.1.2 Prototype

```
void MAC_CbackEvent(macCbackEvent_t *pData)
```

7.1.1.3 Parameter Details

Parameters are passed using the following union. There is a data type for each event.

```
typedef struct
{
  uint8   event;
  uint8   status;
} macEventHdr_t;

typedef union
{
  macEventHdr_t             hdr;
  macMlmeAssociateInd_t     associateInd;
  macMlmeAssociateCnf_t     associateCnf;
  macMlmeDisassociateInd_t  disassociateInd;
  macMlmeDisassociateCnf_t  disassociateCnf;
  macMlmeBeaconNotifyInd_t  beaconNotifyInd;
  macMlmeOrphanInd_t        orphanInd;
  macMlmeScanCnf_t          scanCnf;
  macMlmeStartCnf_t         startCnf;
  macMlmeSyncLossInd_t      syncLossInd;
  macMlmePollCnf_t          pollCnf;
  macMlmeCommStatusInd_t    commStatusInd;
  macMlmePollInd_t          pollInd;
  macMcpsDataCnf_t          dataCnf;
  macMcpsDataInd_t          dataInd;
  macMcpsPurgeCnf_t         purgeCnf;
} macCbackEvent_t;
```

The `hdr.event` field is set to the following values:

NAME	DESCRIPTION
MAC_MLME_ASSOCIATE_IND	Associate indication.
MAC_MLME_ASSOCIATE_CNF	Associate confirm.
MAC_MLME_DISASSOCIATE_IND	Disassociate indication.
MAC_MLME_DISASSOCIATE_CNF	Disassociate confirm.
MAC_MLME_BEACON_NOTIFY_IND	Beacon notify indication.
MAC_MLME_ORPHAN_IND	Orphan indication.
MAC_MLME_SCAN_CNF	Scan confirm.
MAC_MLME_START_CNF	Start confirm.
MAC_MLME_SYNC_LOSS_IND	Sync loss indication.
MAC_MLME_POLL_CNF	Poll confirm.
MAC_MLME_COMM_STATUS_IND	Communication status indication.
MAC_MLME_POLL_IND	Poll indication.
MAC_MCPS_DATA_CNF	Data confirm.
MAC_MCPS_DATA_IND	Data indication.
MAC_MCPS_PURGE_CNF	Purge confirm.
MAC_PWR_ON_CNF	Power on confirm.

7.1.1.4 Return

None.

7.1.2 MAC_CbackCheckPending ( )

7.1.2.1 Description

This callback function returns the number of pending indirect messages queued in the application. Most applications do not queue indirect data and can simply always return zero. The number of pending indirect messages only needs to be returned if `macCfg.appPendingQueue` to TRUE.

The application must implement this function. It is critical that the application implement this function to be as fast as possible; for example, just returning the value of a global variable.

7.1.2.2 Prototype

```
uint8 MAC_CbackCheckPending(void)
```

7.1.2.3 Parameter Details

None.

7.1.2.4 Return

The number of indirect messages queued in the application or zero.

8. Configuration

8.1 Configuration Constants

The MAC uses a structure containing various user-configurable parameters. This structure is called `macCfg` and is in file `mac_cfg.c`. The structure is defined as follows.

```
typedef struct
{
  uint8   txDataMax;
  uint8   txMax;
  uint8   rxMax;
  uint8   dataIndOffset;
  bool    appPendingQueue;
} macCfg_t;
```

NAME	DESCRIPTION	RANGE	DEFAULT
txDataMax	Maximum number of data frames queued in the transmit data queue.	1-255	2
txMax	Maximum number of frames of all types queued in the transmit data queue.	1-255	5
rxMax	Maximum number of frames queued in the receive data queue.	1-255	2
dataIndOffset	Allocate additional bytes in the data indication for application-defined headers.	0-127	0
appPendingQueue	When TRUE, a MAC_MLME_POLL_IND will be sent to the application when a data request command frame is received from another device.	TRUE-FALSE	FALSE

8.2 Compile-Time Configuration

The MAC source code has certain compile-time configurable parameters. To use these parameters most effectively the full source code release of the MAC is recommended.

NAME	DESCRIPTION	RANGE	DEFAULT
MAC_NO_PARAM_CHECK	When FALSE, parameters to API functions are checked as required by [1]. When TRUE the parameter checking code is compiled out. Code size can be reduced slightly by setting to TRUE.	TRUE, FALSE	FALSE
MACNODEBUG	When defined, MAC_ASSERT() run-time error checking is enabled. When undefined this code is compiled out. Code size can be reduced slightly when defined.	N/A	undefined
MAC_RX_ONOFF_DEBUG_LED	When TRUE, an LED is turned on and off when the receiver is turned on and off	TRUE, FALSE	FALSE

9. Scenarios

This section contains scenarios that provide additional information about the operation of the API. Each scenario had a ladder diagram that illustrates how API functions and callback events operate.

9.1 Non beacon-enabled network Start

This scenario shows a PAN coordinator device starting a non beacon-enabled network. It first resets the MAC on device startup. Then it performs an energy detect scan to find an unused channel. Then it performs an active scan to find the channel with the lowest number of active networks. After the scan it sets the MAC attributes it needs to start a network: the short address, beacon payload, and associate permit flag. Then it starts a non beacon-enabled network.

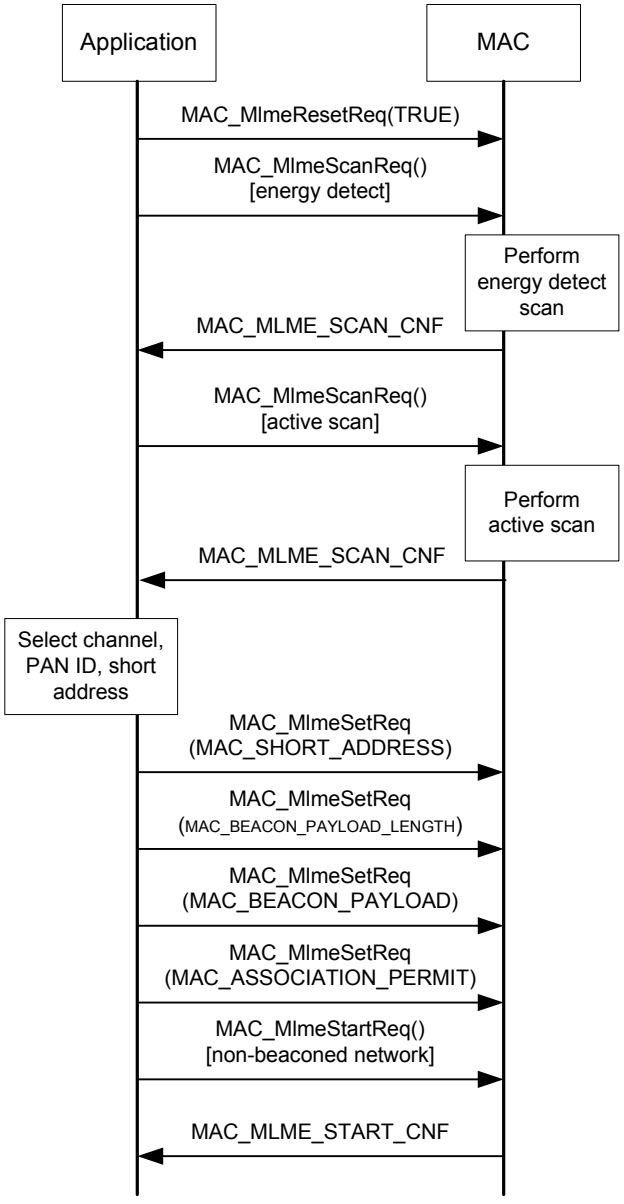

Figure 2. Non beacon-enabled network start.

9.2 Non beacon-enabled network Scan and Associate

This scenario shows a device connecting to a non beacon-enabled network. The device performs an active scan, broadcasting a beacon request on each channel. When the coordinator receives the beacon request it sends a beacon. When the scan is complete the MAC sends a `MAC_MLME_SCAN_CNF` with the PAN descriptors it has received during the scan. The device application examines the PAN descriptors and selects a coordinator. The coordinator application receives a `MAC_MLME_ASSOCIATE_IND` and calls `MAC_MlmeAssociateRsp()` allowing the device to associate. The device application receives a `MAC_MLME_ASSOCIATE_CNF` indicating success. The device application then sets the MAC short address attribute.

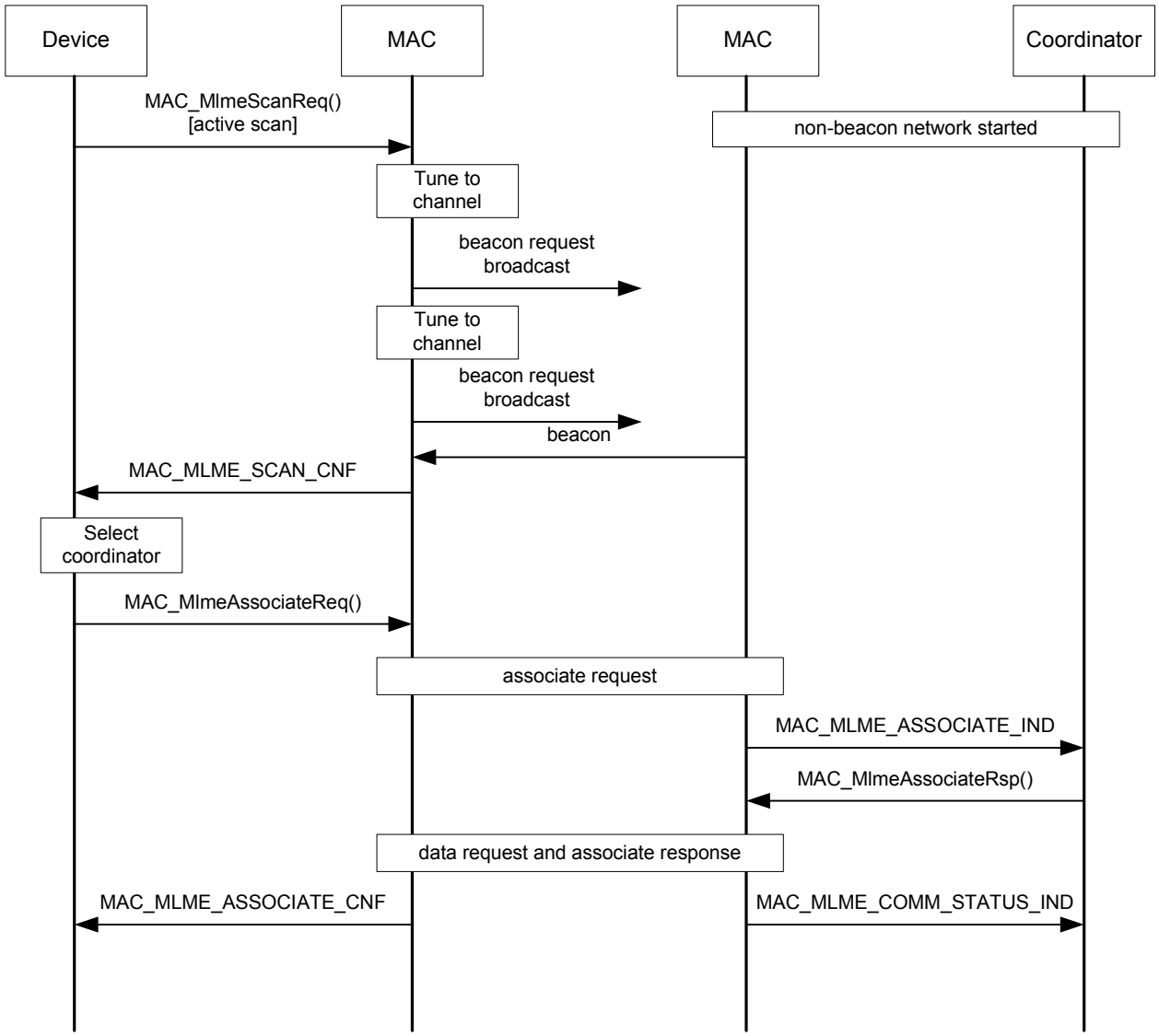

Figure 3. Device connecting to a non beacon-enabled network.

9.3 Beacon-enabled network Scan and Sync

This scenario shows a device synchronizing to a beacon-enabled network. First the device performs a passive scan, tuning to each channel and listening for beacons. When a beacon is received the MAC sends a `MAC_MLME_BEACON_NOTIFY_IND`. When the scan is complete the MAC sends a `MAC_MLME_SCAN_CNF`. The application examines the PAN descriptors, selects a coordinator and sets the coordinator short address, beacon order, and PAN ID. Then the application attempts to synchronize with the coordinator. When the MAC has successfully received the coordinator's beacon and synchronized it sends a `MAC_MLME_BEACON_NOTIFY_IND`. At this point the application sets PIB attribute `MAC_AUTO_REQUEST` to TRUE to prevent further beacon notify indications. After a while the MAC loses synchronization and after four beacon periods it sends a `MAC_MLME_SYNC_LOSS_IND` to the application.

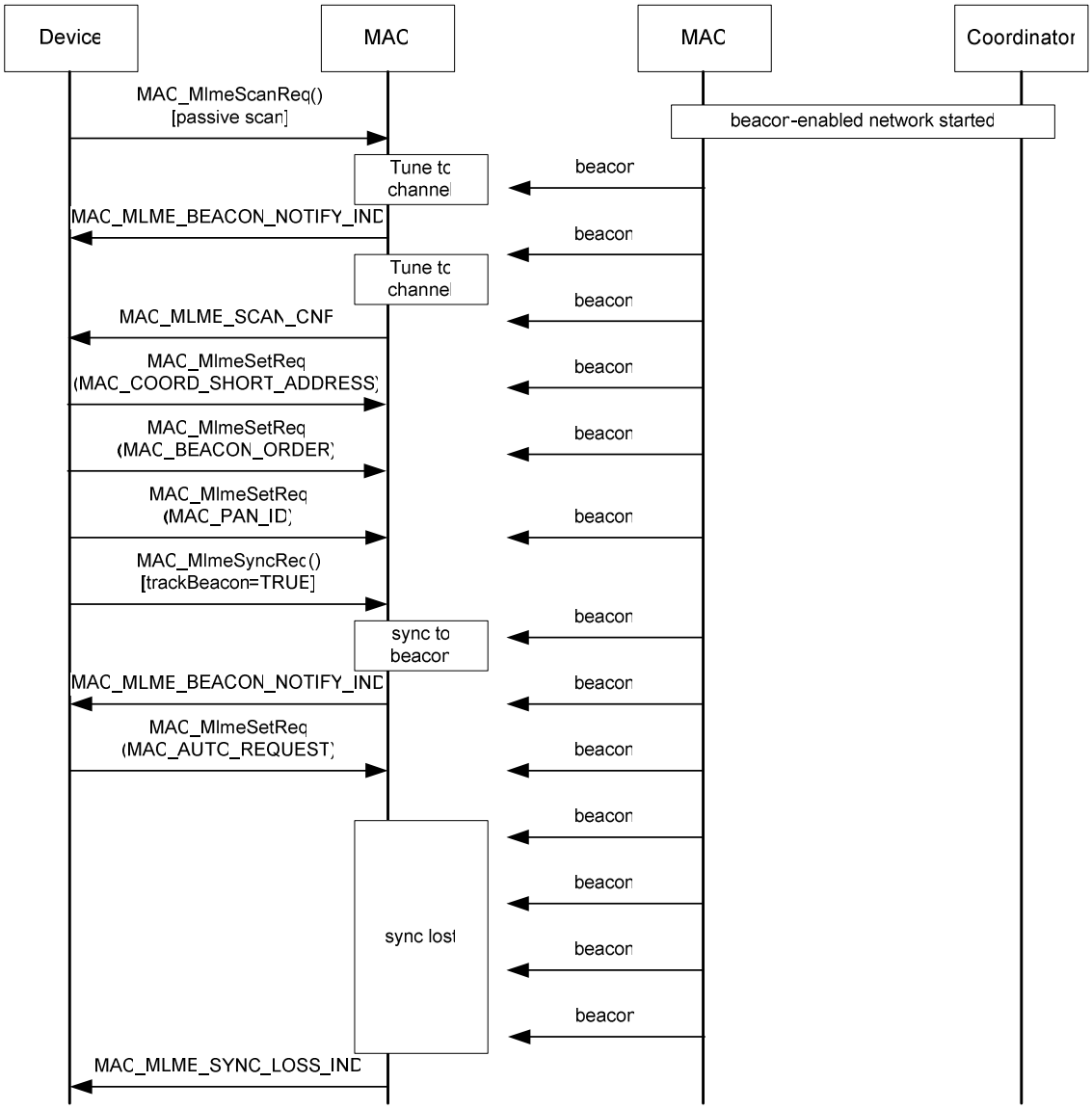

Figure 4. Beacon-enabled network scan and sync.

9.4 Orphan Procedure

This scenario shows a device performing an orphan scan and successful orphan procedure the coordinator. The device performs an orphan scan, broadcasting an orphan notification on each channel. When the orphan notification is received by the coordinator the MAC sends a `MAC_MLME_ORPHAN_IND` to the application. The application allows the orphan and calls `MAC_MlmeOrphanRsp()`. The coordinator sends a coordinator realignment message to the device. Upon receiving the message the orphan procedure is complete and the MAC sends a `MAC_MLME_SCAN_CNF` to the application. The application then sets the short address attribute.

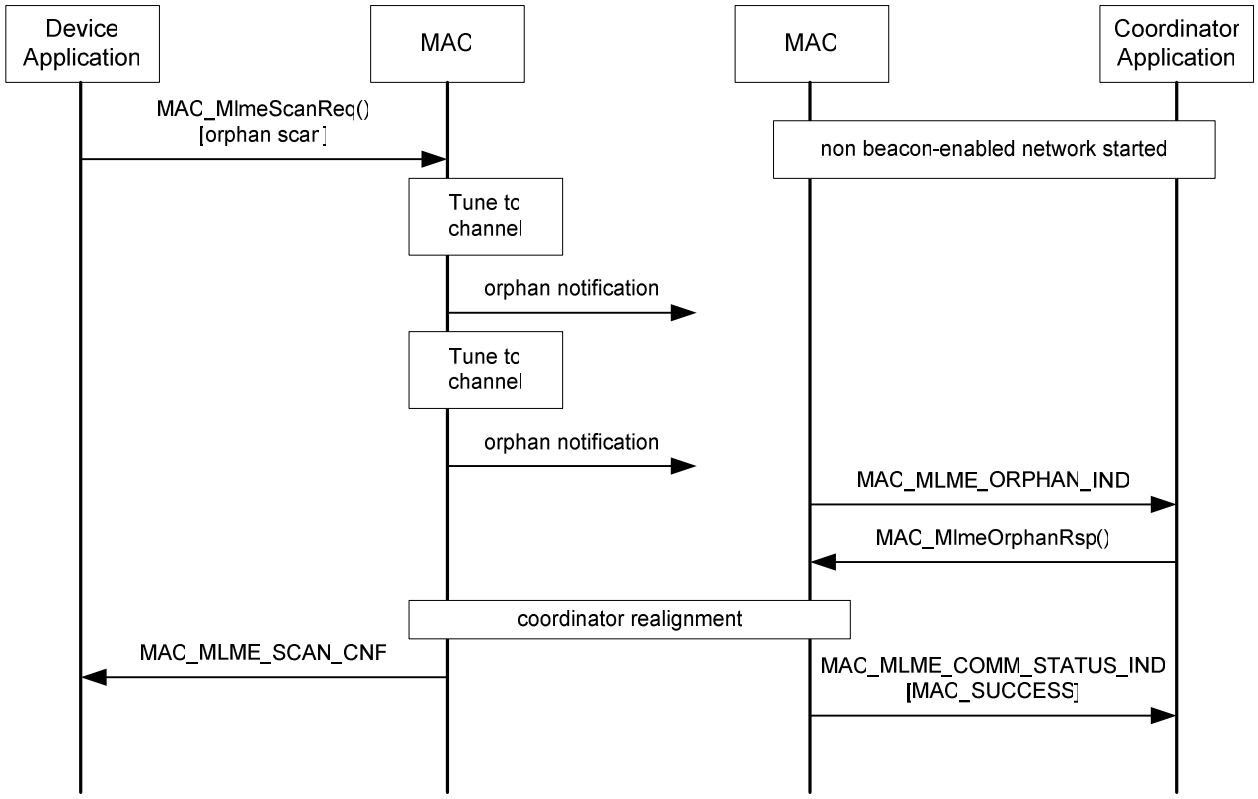

Figure 5. Orphan procedure.

9.5 Direct Data Transactions

This scenario shows various direct data transactions between an FFD device and a coordinator. A basic data transaction is as follows: The device application calls `MAC_McpsDataReq()` to send a data frame. The MAC transmits this frame and receives an acknowledgement. The MAC sends the device application a `MAC_MCPS_DATA_CNF` with status indicating success. On the receiving side, the MAC sends the coordinator application A `MAC_MCPS_DATA_IND` containing the received data frame.

In the next part of the scenario the application calls `MAC_McpsDataReq()` three times without waiting for a `MAC_MCPS_DATA_CNF`. After the third call the MAC sends a `MAC_MCPS_DATA_CNF` with status indicating it is congested and could not send the third data frame. The MAC then transmits the first data frame successfully. Upon receiving the `MAC_MCPS_DATA_CNF`, the application calls `MAC_McpsDataReq()` to send the third frame again. The MAC then attempts to transmit the second frame; the frame is not acknowledged so MAC send a `MAC_MCPS_DATA_CNF` with failure status `MAC_NO_ACK`. The MAC then transmits the third frame successfully.

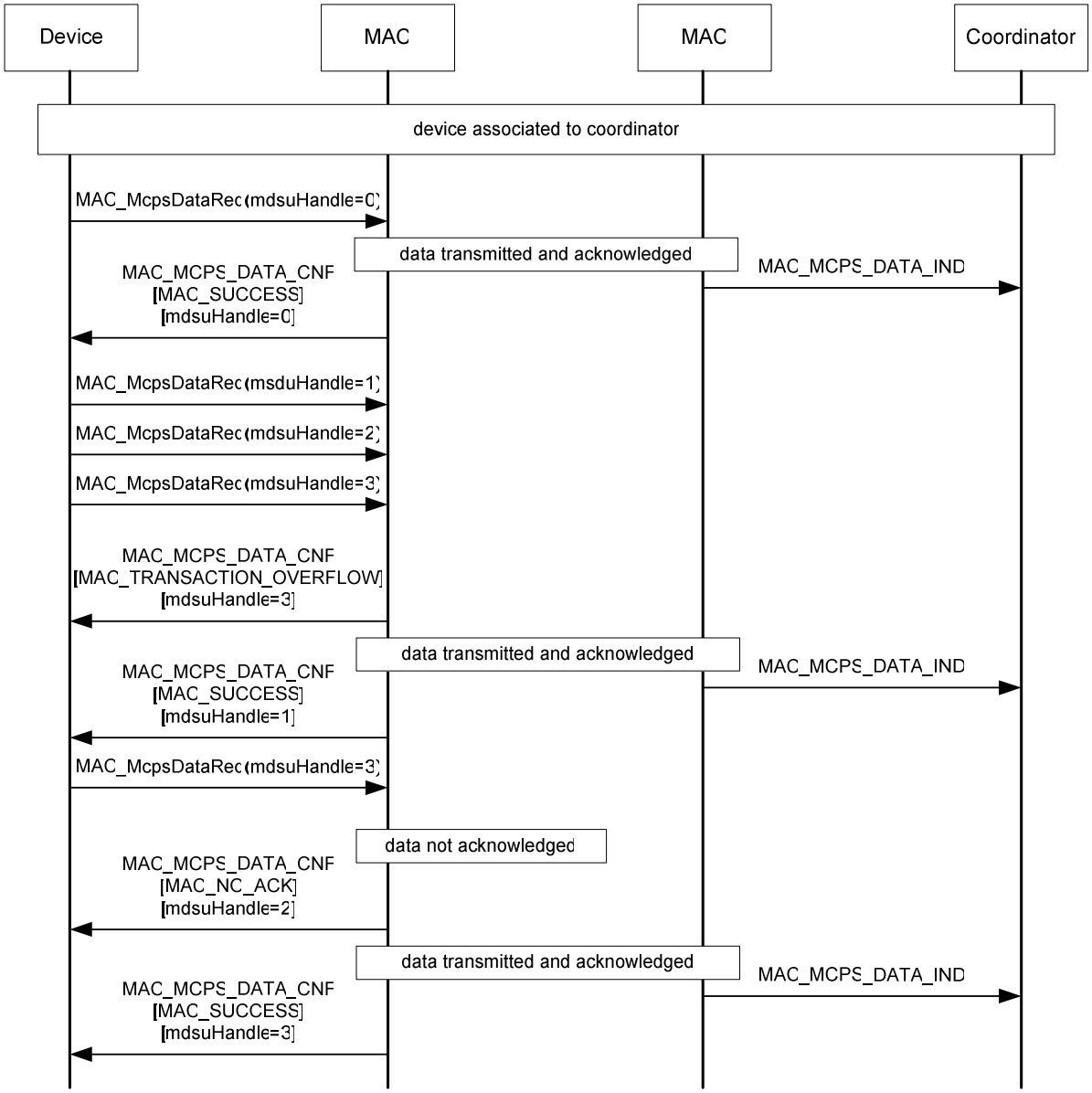

Figure 6. Data transactions.

9.6 Indirect Data Transactions, Data Poll and Purge

This scenario shows data poll and purge operations. The coordinator application calls `MAC_McpsDataReq()` to send data to the device using indirect transmission. Since the data is indirect it is queued inside the MAC. The device calls `MAC_McpsPollReq()` to poll for data. The coordinator sends the data to the device. The MAC on the device side sends a `MAC_MCPS_POLL_CNF` with status indicating success and a `MAC_MCPS_DATA_IND` containing the data packet. The MAC on the coordinator side sends a `MAC_MCPS_DATA_CNF` with status success.

In the next part of the scenario the coordinator calls `MAC_McpsDataReq()` and then `MAC_McpsPurgeReq()`. The data frame queued inside the MAC is successfully purged. The MAC sends a `MAC_MCPS_PURGE_CNF` with status indicating success.

In the next part of the scenario the device polls for data again. There is no data available so the MAC sends a `MAC_MCPS_POLL_CNF` with status `MAC_NO_DATA`.

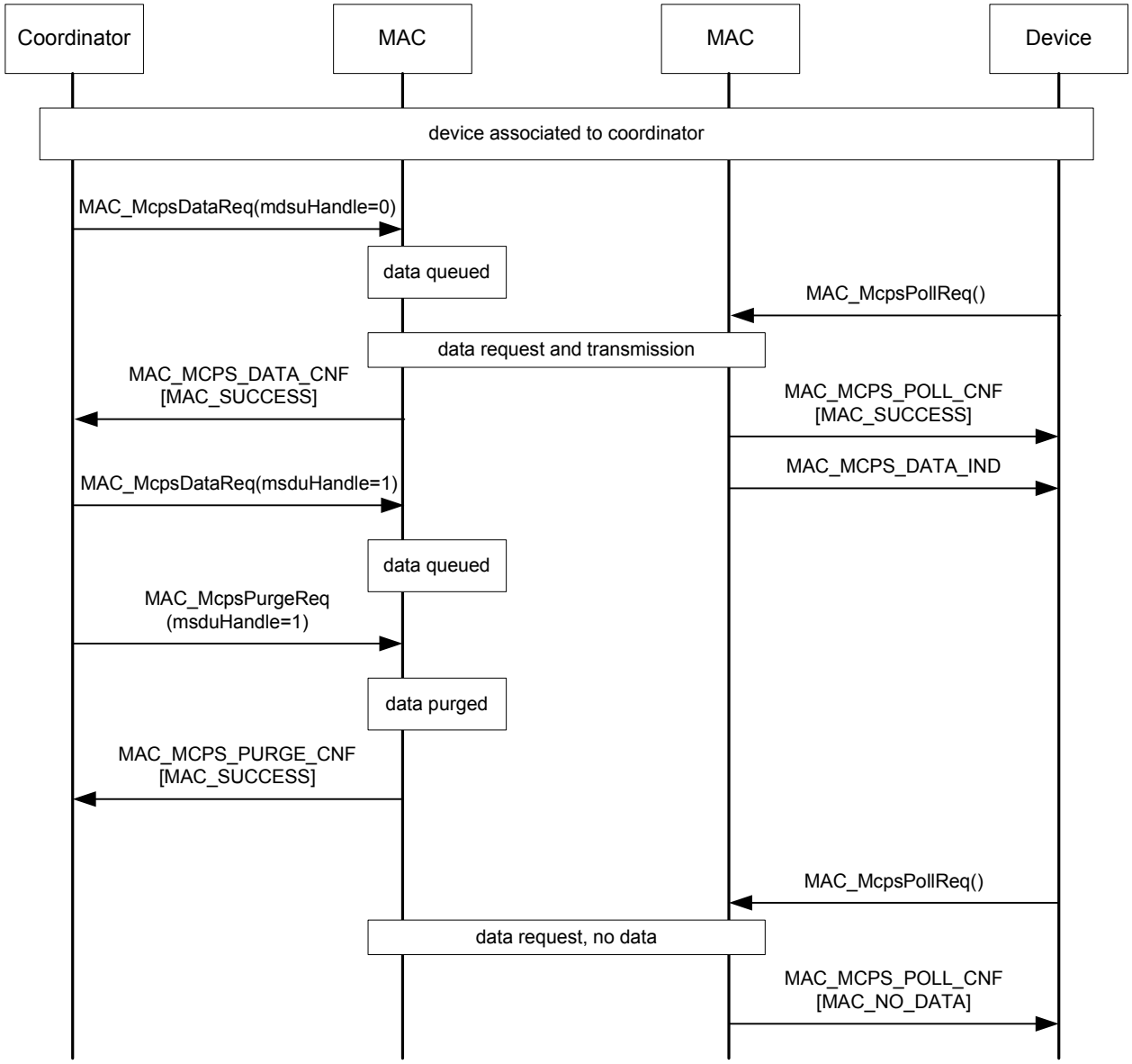

Figure 7. Data poll and purge.