Introduction

The new smart-grid systems and the emerging “Internet of things” concept require advanced web services based on the TCP/IPv6 communication protocol. The TCP/IPv6 stack has been adapted to narrow band communication systems thanks to the work of the 6LowPAN working group of the IETF for narrow band wireless PANs, based on the IEEE-802.15.4 compliant MAC layer.

The same concept can be extended to narrow band wired power line communication networks by implementing an 802.15.4-like MAC layer on a 32-bit microcontroller of the STM32 family, on top of the ST7580 multi mode power line networking SoC. The ST7580 device inside the STarGRID family offers sufficient flexibility to support such protocol and even the STM32F103 MCU has been used for the first firmware release; it can be ported to other devices in the STM32 family.

This document describes the communication protocol and the services provided to the upper layers of the protocol stack.
Contents

1 Overview ............................................................... 5

2 Firmware architecture ............................................. 6

3 Framing ................................................................. 8

4 Service description .................................................. 11
   4.1 Address structure ............................................. 11
   4.2 Data services .................................................. 12
      4.2.1 MCPS-DATA.request .................................... 12
      4.2.2 MCPS-DATA.confirm .................................... 13
      4.2.3 MCPS-DATA.indication .................................. 13
   4.3 Management services ........................................ 14
      4.3.1 MLME-BEACON-NOTIFY.indication ....................... 14
      4.3.2 MLME-GET.request ...................................... 15
      4.3.3 MLME-GET.confirm ..................................... 15
      4.3.4 MLME-SET.request ...................................... 16
      4.3.5 MLME-SET.confirm ..................................... 17
      4.3.6 MLME-RESET.request .................................... 17
      4.3.7 MLME-RESET.confirm ................................... 18
      4.3.8 MLME-SCAN.request .................................... 18
      4.3.9 MLME-SCAN.confirm .................................... 18
      4.3.10 MLME-START.request ................................... 19
      4.3.11 MLME-START.confirm .................................. 19
      4.3.12 MLME-COMM-STATUS.indication ......................... 20
      4.3.13 MLME-SYNC-LOSS.indication ......................... 20

5 Application example ................................................. 22

Appendix A Boot and join sequence ............................... 23

Appendix B Modem setup ............................................. 24

Appendix C PIB attributes ........................................... 25
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>References</td>
<td>27</td>
</tr>
<tr>
<td>7</td>
<td>Revision history</td>
<td>28</td>
</tr>
</tbody>
</table>
List of figures

Figure 1. Firmware architecture ................................................................. 6
Figure 2. PSK frame ..................................................................................... 8
Figure 3. Fields of the MAC frame ................................................................. 8
Figure 4. EVALKITST7580-1 ................................................................. 22
Figure 5. Boot and association of a new node ........................................... 23
1 Overview

The 802.15.4-like communication MAC layer provides services able to build and manage a power line network with a master node referred to as “network coordinator” and several peer nodes called “coordinators”.

Any type of power line network may be created.

In order to start transmissions, a coordinator node needs to join the network; it performs an association procedure called “active scan”.

The protocol has been created for the STM32 family microcontrollers. It assumes a serial communication line (UART) is used to communicate and exchange data with the ST7580 power line modem device.
2 Firmware architecture

Figure 1 shows the firmware architecture:

The ST7580 power line modem provides physical layer services together with some basic data link layer services. The role of each block is as follows:

- **st7580_wrapper**: this module works as a wrapper for any access to the modem. When transmitting, this module sends frames to the modem and manages the local port acknowledgments (UART communication). When receiving, this module analyzes the packet and sends the related event to the right FSM among the MAC machines. The outgoing packets are evaluated by the CSMA/CA JAS anti-collision module in order to obey the medium access protocol rules; the only exceptions are ACK packets and all the segments of a series, excluding the first.

- **MAC**: this is a set of modules which provides all the functionality of the data link layer. Here, several FSMs work together to execute requests issued from the higher layer and manage the data gathered by the modem from the power line (such as other nodes in the network in a neighbor table, the quality level of the medium and internal data structures).

- **MacSapInterface**: this module provides a unique, clean interface to the whole set of MAC services including request, confirmation and indication primitives. This modules is the interface for the application layer which is built upon the power line network device.

Table 1 shows the source files involved in any block.

<table>
<thead>
<tr>
<th>Block</th>
<th>Modules</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>st7580_wrapper</td>
<td>ST7580_Layer\st7580_wrapper (c/h)</td>
<td>Very low-level modules.</td>
</tr>
<tr>
<td></td>
<td>ST7580_Layer\st7580_serial20 (c/h)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BSP\STM32_EVAL\st7580_stm32_bsp (c/h)</td>
<td></td>
</tr>
</tbody>
</table>
### Table 1. Module details

<table>
<thead>
<tr>
<th>MAC</th>
<th>Source Files</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MAC\MacConstants.h</td>
</tr>
<tr>
<td></td>
<td>MAC\MacEnumeration.h</td>
</tr>
<tr>
<td></td>
<td>MAC\MacFrameCreation\Handling (c/h)</td>
</tr>
<tr>
<td></td>
<td>MAC\MacFSM* (c/h)</td>
</tr>
<tr>
<td></td>
<td>MAC\MacMessageCatalog.h</td>
</tr>
<tr>
<td></td>
<td>MAC\MacNeighbourTable (c/h)</td>
</tr>
<tr>
<td></td>
<td>MAC\MacPanDescriptorMng (c/h)</td>
</tr>
<tr>
<td></td>
<td>MAC\MacPibAttributes (c/h)</td>
</tr>
<tr>
<td></td>
<td>MAC\MacPibTypes.h</td>
</tr>
<tr>
<td></td>
<td>MAC\MacSapQueue (c/h)</td>
</tr>
<tr>
<td></td>
<td>DataLinkLayer\channel_access (c/h)</td>
</tr>
<tr>
<td></td>
<td>DataLinkLayer\csmaca (c/h)</td>
</tr>
<tr>
<td></td>
<td>DataLinkLayer\timers (c/h)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MacSapInterface</th>
<th>Source Files</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MAC\MacSapInterface (c/h)</td>
</tr>
</tbody>
</table>

The source files in DataLinkLayer\subfolder are dedicated to the CSMA/CA JAS access protocol; this can be enabled at build time by declaring a special macro `ENABLE_CSMA_CA` (default). If this macro is not declared, no access protocol is used.

Applications may be built using the API these modules provide.
3 Framing

The adopted modulation is an X-PSK and the physical frame is the typical PSK frame, shown in Figure 2.

**Figure 2. PSK frame**

Where:
- Preamble: 32 bit (this field is detected as start-of-frame signal by the receiver)
- Unique word: 32 bit (fixed content - the receiver evaluates SNR of the medium depending on the perceived quality of this field)
- Mode: 8 bit (this field specifies the modulation scheme for the incoming payload – in this protocol version, it is always set to B-PSK)
- P_SDU Length: 8 bit (length of the P_SDU field in bytes)
- P_SDU: variable field, from 0 up to 248 bytes (this is the actual frame payload)

The P_SDU field contains the MAC PDU with the structure shown in figure 3.

**Figure 3. Fields of the MAC frame**

The content of each field is detailed below.

**Segment control header**

This field stores useful data for segmentation and reassembly routines (**Table 2**).

**Table 2. Fields of the segment control header**

<table>
<thead>
<tr>
<th>Field</th>
<th>Byte</th>
<th>Used bits</th>
<th>Bit length</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES</td>
<td>0</td>
<td>7-2</td>
<td>6</td>
<td>Reserved</td>
</tr>
<tr>
<td>CAP</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Channel Access Priority (1: high, 0: normal)</td>
</tr>
<tr>
<td>LSF</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Last Segment Flag (1: yes, 0: no)</td>
</tr>
<tr>
<td>SC</td>
<td>1</td>
<td>7-2</td>
<td>6</td>
<td>Segment Count</td>
</tr>
<tr>
<td>SL[9-8]</td>
<td>1</td>
<td>1-0</td>
<td>2</td>
<td>Segment Length</td>
</tr>
<tr>
<td>SL[7-0]</td>
<td>2</td>
<td>7-0</td>
<td>8</td>
<td>Segment Length</td>
</tr>
</tbody>
</table>
Note that the SL field stores the length of a generic segment in a series, not the actual segment. Therefore, when sending a 140 byte packet, while the actual segment size is 60 bytes, this field is set to 60 in segment #1, then 60 in segment #2 and 60 again in segment #3, even if this last segment only holds the last 20 bytes.

**Frame control header**

This field is used for specifying the frame type and the addressing modes (table 5).

<table>
<thead>
<tr>
<th>Field</th>
<th>Byte</th>
<th>Used bits</th>
<th>Bit length</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame type</td>
<td>0</td>
<td>0-2</td>
<td>3</td>
<td>Frame type (see notes)</td>
</tr>
<tr>
<td>Security enabled</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>Always zero</td>
</tr>
<tr>
<td>Frame pending</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>Always zero</td>
</tr>
<tr>
<td>Ack request</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>Acknowledgment requested</td>
</tr>
<tr>
<td>PAN ID compression</td>
<td>0</td>
<td>6</td>
<td>1</td>
<td>Enabled/disabled (see notes)</td>
</tr>
<tr>
<td>Reserved</td>
<td>0, 1</td>
<td>7, 0-1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Dest. addr. mode</td>
<td>1</td>
<td>2-3</td>
<td>2</td>
<td>Addressing mode (see notes)</td>
</tr>
<tr>
<td>Frame version</td>
<td>1</td>
<td>4-5</td>
<td>2</td>
<td>Always zero</td>
</tr>
<tr>
<td>Source addr. mode</td>
<td>1</td>
<td>6-7</td>
<td>2</td>
<td>Addressing mode (see notes)</td>
</tr>
</tbody>
</table>

The “frame type” field is set to one of the allowed values:
- 000 = Beacon
- 001 = Data frame
- 010 = Acknowledgment
- 011 = MAC Command frame
- 100-111 are reserved

The “PAN ID compression” field saves (in some cases) two bytes: when both the source address and the destination address are present and the source PAN ID is identical to the destination PAN ID, then this field may be set to one (enabled) and only the destination PAN ID is copied into the frame (source address doesn’t have an explicit PAN ID). In any other case, this field is set to zero.

The “addressing mode” fields may be set to one of the allowed values:
- 00 = address field is not present (that is, no address)
- 01 is reserved
- 10 = address field contains PAN ID and a 16 bit short address
- 11 = address field contains PAN ID and a 64 bit extended address

**Sequence number**

This field is under the MAC control. It specifies the sequence number of the frame, which is useful for matching acknowledgments and distinguishing between subsequent retransmissions. A BSN (beacon sequence number) is used for beacon frames and a DSN (data sequence number) is used for data frames.
Frame payload

This field is used for the MAC payload. Different frames are associated with different cases:

- Beacon frame: the payload is 16-bits long and can be set to one of the following two values:
  - MAC_BEACON_SUPERFRAME_PANCOORD (the sender node is a network coordinator and association is allowed);
  - MAC_BEACON_SUPERFRAME_COORD (the sender node is a coordinator and association is allowed);
- MAC Command frame: the command itself is 8 bits long and the payload may vary. Two different command frames are available:
  - MAC_BEACON_REQ_ID (Beacon Request Command Frame – only the command code is specified, no payload);
  - MAC_IDENTITY_ANNOUNCEMENT (Identity Announcement Command Frame – the command code is followed by the 16-bit short address and then by the 64-bit extended address of the node);
- Acknowledgment: no payload;
- Data frame: any payload.

Frame check sequence

Not used. The physical layer of the ST7580 provides hardware error detection, so this field is not filled with any data.
4 Service description

The firmware offers a complete set of service primitives, as listed in Table 4.

<table>
<thead>
<tr>
<th>Service primitive</th>
<th>request</th>
<th>confirm</th>
<th>indication</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCPS-DATA</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>MLME-BEACON-NOTIFY</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MLME-GET</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>MLME-SET</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>MLME-RESET</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>MLME-SCAN</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>MLME-START</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>MLME-COMM-STATUS</td>
<td></td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>MLME-SYNC-LOSS</td>
<td></td>
<td></td>
<td>yes</td>
</tr>
</tbody>
</table>

4.1 Address structure

The address of a node is a commonly used data structure, so it is useful to acquire it beforehand.

```c
typedef struct {
    uint8_t  Mode;
    uint16_t PanID;
    union {
        uint16_t Short;
        uint8_t  Extended[MAC_ADDR64_SIZE];
    };
} MAC_DeviceAddress_t;
```
where:
- `uint8_t Mode` is the address mode (see notes below table 5)
- `uint16_t PanID` is the 16-bit PAN Identifier
- `uint16_t Short` is the 16-bit short address
- `uint8_t Extended` is the 64-bit extended address (the macro `MAC_ADDR64_SIZE` has value 8)

4.2 Data services

The data services include all the primitives related to data transmission, that is, to MCPS-DATA functions such as “request”, “confirm” and “indication”.

4.2.1 MCPS-DATA.request

```c
uint8_t MAC_McpsDataReq(MAC_McpsDataReq_t *McpsDataReq);
```

This function is called when a data transmission is requested. Input data is a reference to a `MAC_McpsDataReq_t` data structure with the following fields:
- `uint8_t SrcAddrMode` The source address mode, as listed in the notes below table 5.
- `MAC_DeviceAddress_t DstAddr` The destination address.
- `uint16_t MsduLength` The length in bytes of the MSDU. The firmware can handle messages up to `NEXT_HIGHER_LAYER_MAX_PDU_LENGTH` bytes in length through segmentation. The default value for `NEXT_HIGHER_LAYER_MAX_PDU_LENGTH` is 1280.
- `uint8_t *Msdu` Pointer to the MSDU.
- `uint8_t MsduHandle` The handle associated with the data to transmit.
- `uint8_t TxOptions` Special transmission options:
  - `b0`: 0 tx without ACK, 1 tx with ACK
  - `b1`: 0 CAP transmission (fixed)
  - `b2`: 0 direct transmission (fixed)
- `uint8_t SecurityLevel` Always zero.
- `uint8_t QoS` Requested Quality of Service: 1 high, 0 normal
Return values:
- MAC_SAP_ERR_NO_ERROR – operation completed successfully;
- MAC_SAP_ERR_IMPLEMENTATION – no spare room in the event queue; other operations are already being performed;
- MAC_SAP_ERR_INVALID_ADDR_MODE – invalid source address mode or destination address mode.

4.2.2 MCPS-DATA.confirm

```c
uint8_t MAC_McpsDataConfirm(MAC_McpsDataConfirm_t *McpsDataConfirm);
```

This “confirm” function checks for the result of the last data transmission request. The input object is a pointer to a MAC_Mcps_DataConfirm_t data structure consisting of the following fields:
- uint8_t ErrorCode
  Error code of the MCPS-DATA.confirm operation – this field is never used in the current firmware version.
- uint8_t MsduHandle
  The handle specified in the MCPS-DATA.request.
- uint8_t Status
  Status of the data transfer operation.
- uint32_t Timestamp
  Time at which the data transfer has been completed. Note: this is the device internal time which counts the time in milliseconds following power-on.

Return values:
- MAC_SAP_ERR_NO_ERROR – confirm event retrieved successfully;
- MAC_SAP_ERR_PREVIOUS_REQUEST – no confirm event retrieved yet.

The “status” field may be set to one of the following:
- MAC_SUCCESS – operation completed successfully;
- MAC_INVALID_ADDRESS – both destination and source addresses are missing;
- MAC_INVALID_QOS – a wrong value for QoS is requested (only 0/1 are valid);
- MAC_ADD_EVENT_ERROR – unable to transmit the packet because the transmitting machine is busy;
- MAC_CHANNEL_ACCESS_FAILURE – unable to transmit because the channel is busy;
- MAC_NO_ACK – Ack (when requested) is not received.

4.2.3 MCPS-DATA.indication

```c
uint8_t MAC_McpsDataIndication(MAC_McpsDataIndication_t *McpsDataIndication);
```

This primitive is issued in case of data reception.

Ack packets (when requested) are handled internally by the MAC routines. The application level receives confirmation of proper handling when an “indication” is signaled.

The “indication” signaling is also associated with a positive result from any frame corruption (reserved fields/wrong values) check.
Explanation of MAC_McpsDataIndication_t data structure:

- **uint8_t ErrorCode**
  The error code related to the reception event. Always set to the value `MAC_SAP_ERR_NO_ERROR`.
- **MAC_DeviceAddress_t SrcAddr**
  The source address.
- **MAC_DeviceAddress_t DstAddr**
  The destination address. Note that this address has already been checked, so this “indication” signal means the packet is in the correct desired destination node.
- **uint16_t MSDULen**
  Length in bytes of the packet.
- **uint8_t MSDU[NEXT_HIGHER_LAYER_MAX_PDU_LENGTH]**
  The actual MSDU.
- **uint8_t msduLinkQuality**
  SNR value detected during reception of this packet. In case of segmented reception, this value derives from the latest received segment.
- **uint8_t DSN**
  Data Sequence Number. The sequence number identification of a single transmission.
- **uint32_t TimeStamp**
  Time of completion of the reception operation.
- **uint8_t SecurityLevel**
  Information about security setting (cryptography) of the frame. Not used in the current version of the firmware.
- **uint8_t QoS**
  Requested quality of service set by the sender (0: normal quality, 1: high quality).

Return values:

- **MAC_SAP_ERR_NO_ERROR** – the function returned a valid “indication” event;
- **MAC_SAP_ERR_NO_INDICATION_PRESENT** – no “indication” present, that is, no packet received at the moment.

### 4.3 Management services

The following services involve the MLME (MAC subLayer Management Entity) and all the internal MAC management.

#### 4.3.1 MLME-BEACON-NOTIFY.indication

```c
uint8_t MAC_MlmeBeaconNotify(MAC_MlmeBeaconNotify_t *MlmeBeaconNotify);
```

This primitive is issued when a node receives a beacon. The fields of the `MAC_MlmeBeaconNotify_t` data structure are set as follows:

- **uint8_t ErrorCode**
  The error code returned by the reception operation.
- **MAC_PanDescriptor_t PanDescriptor**
  The data structure filled with data from the sender coordinator node.
The return values:
- MAC_SAP_ERR_NO_ERROR – a valid “indication” was retrieved;
- MAC_SAP_ERR_NOINDICATION_PRESENT – no “indication” event is present.

The ErrorCode field is always set with value MAC_SAP_ERR_NO_ERROR.

The MAC_PanDescriptor_t structure has the following fields:
- MAC_DeviceAddress_t CoordAddress
  Address of the sender coordinator.
- uint16_t SuperFrameSpec
  The two-byte SuperFrameSpecification received as payload of the beacon.
- uint8_t GTSPermit
  Permission for GTS (guaranteed time slot). Not used in this protocol version; it is always zero.
- uint8_t LinkQuality
  Signal quality at reception of the beacon, that is, the SNR detected during decoding.
- uint32_t Timestamp
  Arrival time of the beacon.
- uint8_t SecurityFailure
  This field signals problems during decryption of the frame. Not used in this protocol version, always set to MAC_SUCCESS.
- uint8_t SecurityLevel
  Cryptographic protection level of the Beacon frame. Not used in this protocol version.

4.3.2 MLME-GET.request

uint8_t MAC_MlmeGetReq(MAC_MlmeGetReq_t *MlmeGetReq);

This primitive requests a “get” (read) operation on the MAC internal database, PIB (PAN Information Base), providing the object that the function must read.

Fields of the MAC_MlmeGetReq structure:
- uint32_t PIBAttribute
  Integer ID of the attribute to read. See Appendix 3 for Attribute list.
- uint8_t PIBAttributeIndex
  Index used in case of list-attribute.

Return values:
- MAC_SAP_ERR_NO_ERROR – the primitive was issued with no errors;
- MAC_SAP_QUEUE_ERRFULL – unable to issue the event to the FSM.

The only list attribute managed by the MAC is the neighbor table, which contains information about any known node of the network. Its ID is MAC_NEIGHBORTABLE_ID.

4.3.3 MLME-GET.confirm

uint8_t MAC_MlmeGetConfirm(MAC_MlmeGetConfirm_t *MlmeGetConfirm);

This function checks whether a “confirm” event has been issued following a previous GET.request and returns the attribute value.
Explanation of the MAC_MlmeGetConfirm data structure:

- **uint8_t ErrorCode**
  Result of the operation, always MAC_SAP_ERR_NO_ERROR.

- **uint8_t State**
  State flag of the GET operation; it may have one of the following values:
  MAC_SUCCESS for a successful read, MAC_UNSUPPORTED_ATTRIBUTE when the requested attribute is not found or MAC_INVALID_INDEX when a list attribute is specified but the wrong index is provided.

- **uint32_t PIBAttribute**
  ID of the attribute

- **uint8_t PIBAttributeIndex**
  Attribute index for a GET operation on a list attribute.

- **uint8_t PIBAttributeValue[SAP_MAX_ATTRIBUTEVALUEGET_LEN]**
  The value retrieved by the GET operation. The buffer is SAP_MAX_ATTRIBUTEVALUEGET_LEN bytes long (default 20), although 13 are used at most.

- **uint8_t PIBAttributeValueLen**
  Length in bytes of the attribute.

Return values:

- **MAC_SAP_ERR_NO_ERROR** – a valid “confirm” event was copied into the MAC_MlmeGetConfirm data structure;
- **MAC_SAP_ERR_PREVIOUS_REQUEST** – no “confirm” event was found.

**Note:** The PIBAttributeValue buffer has different contents depending on the actual attribute. For a read from the Neighbor Table, it is structured as a small set of integer values related to a specified node, in the following sequence:

- 2 bytes for the actual PAN ID;
- 2 bytes for the actual short address (optional);
- 8 bytes for the extended address (optional);
- 1 byte for the age of the entry (in minutes).

This structure is therefore 5 bytes long (when only the short address is present) or 11 bytes long (when only the extended address is present) or 13 bytes long (when both short and extended address are present). In order to distinguish between these cases, refer to the PIBAttributeValueLen value beforehand.

For integer values, the structure is simpler (1, 2, 4 bytes).

When reading the extended address of the current node, the PIBAttributeValue is set with a pointer (32-bit value) to an 8-byte buffer with the long address.

### 4.3.4 MLME-SET.request

```c
uint8_t MAC_MlmeSetReq(MAC_MlmeSetReq_t *MlmeSetReq);
```
This function is called when a SET.request must be issued. The parameter contains the attribute identifier and its new value. The MAC_MlmeSetReq_t data structure fields are:

- `uint32_t PIBAttribute`
  The integer identifier of the attribute to be set. See Appendix 3 for Attribute list.

- `uint8_t PIBAttributeIndex`
  The index for a list attribute.

- `union PIBAttributeValue`
  This field has one of the following values: value8 (unsigned 8-bit value), value16 (unsigned 16-bit value), value32 (unsigned 32-bit value), buffer (a generic buffer – its length is SAP_MAX_ATTRIBUTEVALUESET_LEN bytes, defaulted to 4).

- `uint8_t PIBAttributeValueLen`
  Length in bytes of the value stored in the PIBAttributeValue field.

Return values:

- `MAC_SAP_ERR_NO_ERROR` – the “request” primitive has been successfully issued to the MAC;
- `MAC_SAP_QUEUE_ERRFULL` – the FSM has no further room for a new SET.request event.

4.3.5 MLME-SET.confirm

`uint8_t MAC_MlmeSetConfirm(MAC_MlmeSetConfirm_t *MlmeSetConfirm);`

This function checks for a SET confirm event. The MAC_MlmeSetConfirm_t data structure has the following fields:

- `uint8_t ErrorCode`
  The error code returned from the “confirm” primitive; it is always `MAC_SAP_ERR_NO_ERROR`.

- `uint8_t State`
  The state of the SET operation. It may have one of the following values: `MAC_SUCCESS` (when there are no errors), `MAC_UNSUPPORTED_ATTRIBUTE` (when the attribute is not found), `MAC_READ_ONLY` (the requested attribute is read-only).

- `uint32_t PIBAttribute`
  The 32 bit identifier of the specified attribute.

- `uint8_t PIBAttributeIndex`
  The index for a list attribute.

Return values:

- `MAC_SAP_ERR_NO_ERROR` – the function call returns a valid “confirm” event;
- `MAC_SAP_ERR_PREVIOUS_REQUEST` – no “confirm” event was found.

4.3.6 MLME-RESET.request

`uint8_t MAC_MlmeResetReq(MAC_MlmeResetReq_t *MlmeResetReq);`

This function requests a reset of the MAC layer. The MAC_MlmeResetReq_t data structure has a single field:

- `uint8_t SetDefaultPIB`
  This flag tells the engine whether or not to reset the PIB during the software reset. Set it to 1 to reset or 0 to leave it with latest values.
The reset operation always involves a hardware reset of the ST7580 modem. The PANDescriptorList and the neighbor table structures are also cleaned.

Return values:
- MAC_SAP_ERR_NO_ERROR – the request was successfully issued;
- MAC_SAP_ERR_IMPLEMENTATION – the Reset FSM was unable to accept a new request.

4.3.7 MLME-RESET.confirm

uint8_t MAC_MlmeResetConfirm(MAC_MlmeResetConfirm_t *MlmeResetConfirm);

This function checks for a “confirm” for a previous RESET.request primitive. The MAC_MlmeResetConfirm_t data structure has the following fields:
- uint8_t ErrorCode
  Result of the “confirm” event; it is always set to MAC_SAP_ERR_NO_ERROR.
- uint8_t Status
  Status of the reset operation; it always has the value MAC_SUCCESS.

Return values:
- MAC_SAP_ERR_NO_ERROR – the “confirm” event was retrieved successfully;
- MAC_SAP_ERR_PREVIOUS_REQUEST – no “confirm” event.

4.3.8 MLME-SCAN.request

uint8_t MAC_MlmeScanReq(MAC_MlmeScanReq_t *MlmeScanReq);

When a node starts up, the next layer above MAC calls the active scan procedure in order to check the actual network status. This function issues the necessary primitive and other nodes, if any, send their beacons. See Appendix A for the complete description of the join procedure. The parameters in the MAC_MlmeScanReq_t are:
- uint8_t ScanType
  Select the scan type. Only Active Scan is allowed; set this to the value MAC_ACTIVE_SCAN.
- uint32_t ScanChannels
  Used for selecting channels to check during the scan operation. Not used in this protocol version.
- uint8_t ScanDuration
  Duration in seconds for the whole scan operation (max. is 60, a zero setting is not allowed).
- uint8_t ChannelPage
- uint8_t SecurityLevel
  These parameters are not used in this version of the software.

Return values:
- MAC_SAP_ERR_NO_ERROR – the “request” event was successfully issued to the Scan FSM;
- MAC_SAP_ERR_IMPLEMENTATION – unable to send the request to the scan FSM.

4.3.9 MLME-SCAN.confirm

uint8_t MAC_MlmeScanConfirm(MAC_MlmeScanConfirm_t *MlmeScanConfirm);
This function checks for a “confirm” event after a previous scan request. The MAC_MlmeScanConfirm_t data structure has the following fields:

- **MAC_PanDescriptorList_t PanDescriptorList**
  This structure consists of an unsigned byte, PanDescriptorCount, which counts how many Beacon frames have been received, and an array of MAC_PanDescriptor_t PanDescriptor; these have the structure shown in Chapter 4.3.1.

- **uint8_t ErrorCode**
  Error code of the “confirm” event; it is always set to MAC_SAP_ERR_NO_ERROR.

- **uint8_t Status**
  Status of the Scan operation; it may have one of the following values: MAC_SUCCESS (successfully completed), MAC_SCAN_IN_PROGRESS (another Scan sequence was already pending), MAC_INVALID_PARAMETER (a wrong Scan value was specified for the scan type and/or duration), MAC_ADD_EVENT_ERROR (unable to transmit the Beacon Request Command frame), or any of the error codes returned by the transmission FSM (listed in Chapter 4.2.2).

- **uint8_t ScanType**
  Always set to MAC_ACTIVE_SCAN.

Return values:

- **MAC_SAP_ERR_NO_ERROR** – the “confirm” data was successfully retrieved;
- **MAC_SAP_ERR_PREVIOUS_REQUEST** – no “confirm” event available.

### 4.3.10 MLME-START.request

```c
uint8_t MAC_MlmeStartReq(MAC_MlmeStartReq_t *MlmeStartReq);
```

With this function, a node can set the PAN ID and its role in the network (coordinator/PAN coordinator). The parameters are found in the MAC_MlmeStartReq_t structure:

- **uint8_t CoordRole**
  This flag controls whether the node is a PAN Coordinator (set this field to DEVICE_TYPE_PAN_COORDINATOR) or a Coordinator (set this field to DEVICE_TYPE_COORDINATOR)

- **uint16_t PanId**
  The 16 bit PAN identifier. Note that, although the application may choose this identifier randomly, the protocol requires that it is logically and-ed with 0xFCFF.

Return value:

- **MAC_SAP_ERR_NO_ERROR** – the request was issued to the MAC.

### 4.3.11 MLME-START.confirm

```c
uint8_t MAC_MlmeStartConfirm(MAC_MlmeStartConfirm_t *MlmeStartConfirm);
```

When called, this function checks for a “confirm” event relative to a previous start request.
The results are stored into the MAC_MlmeStartConfirm_t structure:

- **uint8_t ErrorCode**
  
  Error code from the “confirm” retrieval operation; it is always set to MAC_SAP_ERR_NO_ERROR.

- **uint8_t Status**
  
  Status code of the Start operation; it may have the values MAC_SAP_ERR_NO_ERROR (Start sequence completed successfully) or MAC_INVALID_PARAMETER (the specified PAN ID is invalid).

Return values:

- MAC_SAP_ERR_NO_ERROR – a valid “confirm” event is present;
- MAC_SAP_ERR_PREVIOUS_REQUEST – no “request” was previously issued.

### 4.3.12 MLME-COMM-STATUS.indication

This function checks for a communication status “indication” event from the MAC, that is, a detailed report about the last communication event. The MAC_MlmeCommStatusIndication_t data structure has the following fields:

- **uint8_t ErrorCode**
  
  The error code of the “confirm” event check; it is always set to MAC_SAP_ERR_NO_ERROR.

- **uint16_t PanId**
  
  This is the PAN identifier of the source node.

- **MAC_DeviceAddress_t SrcAddr**

- **MAC_DeviceAddress_t DstAddr**

  The addresses of both source and destination nodes.

- **uint8_t Status**

  Last result of the communication channel. May be one of the following values: MAC_COUNTER_ERROR (problem during reassembly process due to wrong segment received), MAC_NO_ACK (after transmission, an ack was expected but never received), MAC_CHANNEL_ACCESS_FAILURE (the medium was detected as busy too many times, transmission aborted).

- **uint8_t SecurityLevel**

  Code related to cryptography of the message. Not used in this version of the software.

Return values:

- MAC_SAP_ERR_NO_ERROR – an “indication” event was successfully retrieved.
- MAC_SAP_ERR_NOINDICATION_PRESENT – no “indication” was found.

### 4.3.13 MLME-SYNC-LOSS.indication

This function checks for a “lost” event from the MAC, that is, a report about a received message that could not be reassembled due to lost segments.

The MAC_MlmeSyncLossIndication_t data structure has the following fields:

- **uint8_t ErrorCode**

  Error code from the “confirm” retrieval operation; it is always set to MAC_SAP_ERR_NO_ERROR.

- **uint16_t PanId**

  This is the PAN identifier of the source node.

- **MAC_DeviceAddress_t SrcAddr**

- **MAC_DeviceAddress_t DstAddr**

  The addresses of both source and destination nodes.

- **uint8_t Status**

  Last result of the communication channel. May be one of the following values: MAC_COUNTER_ERROR (problem during reassembly process due to wrong segment received), MAC_NO_ACK (after transmission, an ack was expected but never received), MAC_CHANNEL_ACCESS_FAILURE (the medium was detected as busy too many times, transmission aborted).

- **uint8_t SecurityLevel**

  Code related to cryptography of the message. Not used in this version of the software.

Return values:

- MAC_SAP_ERR_NO_ERROR – an “indication” event was successfully retrieved.
- MAC_SAP_ERR_NOINDICATION_PRESENT – no “indication” was found.
This primitive is issued to the next layer above when an overlapping PAN is detected – this case is called PAN ID conflict. The structure of the MAC_MlmeSyncLossIndication_t block is as follows:

- `uint8_t ErrorCode`
  Error code related to the “indication” event; it is always set to `MAC_SAP_ERR_NO_ERROR`.
- `uint8_t LossReason`
- `uint16_t PanId`
  The new discovered PAN identifier.

Return values:

- `MAC_SAP_ERR_NO_ERROR` – an “indication” event was successfully retrieved.
- `MAC_SAP_ERR_NOINDICATION_PRESENT` – no “indication” event found.
5 Application example

The protocol library is delivered with a simple application example running on the EVALKITST7580-1, shown in the following picture.

Figure 4. EVALKITST7580-1

The application simulates a simple data collection operation from “sensor” nodes to a data concentrator, and a command transmission operation from the data concentrator to the “sensor” nodes.

It functions along the following description, using at least two kits for communication evaluation:

Following boot, the board starts the active scan procedure for 10 seconds and, if no beacon are received (because it is alone or it is the first to be powered up), it elects itself as PAN coordinator and starts the MAC as PAN coordinator. During scan procedure the LEDs DL3 and DL5 blink.

If a beacon is received because a PAN coordinator already exists, the board starts the MAC as a normal coordinator with the retrieved PAN_ID and a random local short address.

After the association process finishes, the normal coordinator sends dummy messages to the PAN coordinator periodically. During normal operation of the coordinator, the LED DL5 blinks.

The PAN coordinator maintains a table of 3 nodes from the MAC neighbor table associated with LEDs DL3, DL4 and DL5 of the board; each time it receives a message from one normal coordinator, it toggles the associated LED. If the TEST 1 button of the PAN coordinator board is pushed, a broadcast command message is sent; as soon as the coordinator boards receive the command, they toggle the LED DL4.

For correct operation, a main super loop must call the MAC_FSMHandler function to run all the MAC finite state machines and the MODEM_FSM_Management function to run the STM32-ST7580 local communication finite state machine.
Appendix A  Boot and join sequence

Figure 5 shows the boot of a new node and the association procedure. This includes the actual scan of the network, called “active scan”.

Figure 5. Boot and association of a new node

The actions performed by the node are:

- **MLME-RESET**: hardware and software reset of the device (modem and MAC).
- **MLME-SCAN**: scan of the network nodes (if any) through the beacon request command frame. Note that the duration is set by the caller.
- Reception of beacon frames sent by other nodes.
- **MLME-SET**: setting of the attribute MAC_SHORTADDRESS_ID (also known as the short address of the device) and the node activity between the PAN coordinator and coordinator. Note that for the node to be a PAN Coordinator, the short address must be set to 0 (zero).
- **MLME-START**: start of the node transmissions. In this step, the node sets the PAN identifier and sends the identity announcement command frame with its own parameters such as associated PAN, short address and extended address.
Appendix B  Modem setup

The main program can customize the modem configuration in order to set frequencies, reception mode, modulation scheme and the like.

```c
void DL_LoadStartupConfig(DL_OBJ_T *DataLink, uint32_t freqH, uint32_t freqL, uint8_t mod, uint8_t channel_config);
```

This function reads the default settings of the modem, copies them into the DataLink data structure and then customizes those values with the new ones provided by the caller. They are:

- `uint32_t freqH`
  The high reception frequency for the modem
- `uint32_t freqL`
  The low reception frequency for the modem.
- `uint8_t mod`
  Modulation scheme selection; this can be set to one of the following values:
  - MODEM_MOD_BPSK
  - MODEM_MOD_QPSK
  - MODEM_MOD_8PSK
  - MODEM_MOD_BPSK_CODED
  - MODEM_MOD_QPSK_CODED
  - MODEM_MOD_BPSK_CODED_PNA
- `uint8_t channel_config`
  Select between single channel reception (using either high or low frequency) and dual channel reception. This can be set to MODEM_PHY_CONFIG_RXMODE_DUAL for reception from both channels, MODEM_PHY_CONFIG_RXHIGH_PSK or MODEM_PHY_CONFIG_RXLOW_PSK for receiving frames from a high frequency channel or from a low frequency channel.
Appendix C  PIB attributes

The application is able to read and modify MAC internal attributes in order to change software behavior or access other data. Table 5 shows these attributes.

Table 5. List of PIB attributes

<table>
<thead>
<tr>
<th>Identifier</th>
<th>R/W</th>
<th>Length</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC_ACKWAITDURATION_ID (unit is 100 ms)</td>
<td>R/W</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>MAC_ASSOCIATEDPANCOORD_ID</td>
<td>R</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MAC_ASSOCIATIONPERMIT_ID</td>
<td>R</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MAC_MAXBE_ID</td>
<td>R</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>MAC_MAXCSMABACKOFFS_ID</td>
<td>R</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>MAC_MINBE_ID</td>
<td>R</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>MAC_RESPONSEWAITTIME_ID</td>
<td>R</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>MAC_SECURITYENABLED_ID</td>
<td>R</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MAC_SHORTADDRESS_ID</td>
<td>R/W</td>
<td>2</td>
<td>0xFFFF</td>
</tr>
<tr>
<td>MAC_TIMESTAMPSUPP_ID</td>
<td>R</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MAC_HIGHPRIORITYWS_ID</td>
<td>R</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MAC_TXDATAPACKETCOUNT_ID</td>
<td>R/W</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>MAC_RXDATAPACKETCOUNT_ID</td>
<td>R/W</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>MAC_TXCMDPACKETCOUNT_ID</td>
<td>R/W</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>MAC_RXCMDPACKETCOUNT_ID</td>
<td>R/W</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>MAC_CSMAFAILCOUNT_ID</td>
<td>R</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>MAC_CSMACOLLCOUNT_ID</td>
<td>R</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>MAC_BROADCASTCOUNT_ID</td>
<td>R/W</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>MAC_MULTICASTCOUNT_ID</td>
<td>R/W</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>MAC_BADCRCCOUNT_ID</td>
<td>R</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>MAC_MAXORPHANTIMER_ID</td>
<td>R</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>MAC_NEIGHBORTABLE_ID</td>
<td>R</td>
<td>1</td>
<td>N/A</td>
</tr>
<tr>
<td>MAC_PANID_ID</td>
<td>R/W</td>
<td>2</td>
<td>0xFFFF</td>
</tr>
<tr>
<td>MAC_NUMBEROFHOPS_ID</td>
<td>R</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MAC_FREQNOTCHING_ID</td>
<td>R/W</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MAC_BSN_ID</td>
<td>R</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MAC_DSN_ID</td>
<td>R</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MAC_COORDINATOR_TYPE_ID</td>
<td>R/W</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MAC_PROMISCUOUS_MODE</td>
<td>R/W</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MAC_MAXFRAMERETRIES_ID</td>
<td>R/W</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
Table 5. List of PIB attributes (continued)

<table>
<thead>
<tr>
<th>Identifier</th>
<th>R/W</th>
<th>Length</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC_LASTPACKETLENGTH</td>
<td>R/W</td>
<td>1</td>
<td>48</td>
</tr>
<tr>
<td>MAC_EXTENDEDADDRESS_ID (pointer)</td>
<td>R</td>
<td>4</td>
<td>N/A</td>
</tr>
</tbody>
</table>
6 References

- ST7580: datasheet
- UM0932: user manual
- STM32F103 datasheet
- STM32F103 reference manual
- EVALKITST7580-1 data brief
- AN4068
## 7 Revision history

Table 6. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Changes</th>
</tr>
</thead>
</table>
| 04-Jun-2014| 1        | Initial release.
Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale. Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

ST PRODUCTS ARE NOT DESIGNED OR AUTHORIZED FOR USE IN: (A) SAFETY CRITICAL APPLICATIONS SUCH AS LIFE SUPPORTING, ACTIVE IMPLANTED DEVICES OR SYSTEMS WITH PRODUCT FUNCTIONAL SAFETY REQUIREMENTS; (B) AERONAUTIC APPLICATIONS; (C) AUTOMOTIVE APPLICATIONS OR ENVIRONMENTS, AND/OR (D) AEROSPACE APPLICATIONS OR ENVIRONMENTS. WHERE ST PRODUCTS ARE NOT DESIGNED FOR SUCH USE, THE PURCHASER SHALL USE PRODUCTS AT PURCHASER’S SOLE RISK, EVEN IF ST HAS BEEN INFORMED IN WRITING OF SUCH USAGE, UNLESS A PRODUCT IS EXPRESSLY DESIGNATED BY ST AS BEING INTENDED FOR “AUTOMOTIVE, AUTOMOTIVE SAFETY OR MEDICAL” INDUSTRY DOMAINS ACCORDING TO ST PRODUCT DESIGN SPECIFICATIONS. PRODUCTS FORMALLY ESCC, QML OR JAN QUALIFIED ARE DEEMED SUITABLE FOR USE IN AEROSPACE BY THE CORRESPONDING GOVERNMENTAL AGENCY.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

© 2014 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com