**IEEE P802.15**

**Wireless Personal Area Networks**

|  |  |
| --- | --- |
| Project | IEEE P802.15 Working Group for Wireless Personal Area Networks (WPANs) |
| Title | **<MMI and PDE Operation>** |
| Date Submitted | [7 November 2017] |
| Source | [Pat Kinney][<company>][address] | Voice: [ ]Fax: [ ]E-mail: [ ] |
| Re: | TG12 Architecture: PDE and MMI operation |
| Abstract | [Work in Progress – ] |
| Purpose | [Description of what the author wants P802.15 to do with the information in the document.] |
| Notice | This document has been prepared to assist the IEEE P802.15. It is offered as a basis for discussion and is not binding on the contributing individual(s) or organization(s). The material in this document is subject to change in form and content after further study. The contributor(s) reserve(s) the right to add, amend or withdraw material contained herein. |
| Release | The contributor acknowledges and accepts that this contribution becomes the property of IEEE and may be made publicly available by P802.15. |

# Operation Details of ULI Mandatory Elements

As shown in Figure 0.1, the mandatory elements of the ULI are the Protocol Discrimination Entity (PDE), Multiplexed MAC Interface (MMI), Management Protocols Module (MPM), and the Passthru module (PTM).



Figure 0‑1

## PDE Description

### Purpose:

Directs and formats information from protocol module SAP to the appropriate higher layer SAP or to another protocol module SAP as per dictate from the MMI.

### Overview

For frames going to the higher layer, the PDE determines the appropriate SAP for delivery, as determined by the ULI header, removes the ULI header, reconstitutes the appropriate header, and then directs the datagram to the SAP.

For datagrams coming from a higher layer, the PDE determines the SAP to which the datagram is to be sent based upon the configuration of the device as set by the Management Protocols entity, and forwards it to the appropriate SAP.

#### EtherType

Protocol discrimination is based on either EtherTypes or dispatch codes.

EtherType protocol identification values are assigned by the IEEE RA and are used to identify the protocol that is to be invoked to process the user data in the frame. An EtherType is a sequence of 2 octets, interpreted as a 16-bit numeric value with the first octet containing the most significant 8 bits and the second octet containing the least significant 8 bits. Values in the 0–1535 range are not available for use as EtherTypes, rather they designate dispatch codes.

| **EtherType** | **Organization/Address** | **Protocol** |
| --- | --- | --- |
| 0800 | Xerox | IPv4 Internet Protocol Version A Standard for the Transmission of IP Datagrams over Ethernet Networks, RFC-Internet Society, Apr. 1984. http://www.ietf.org/rfc/rfc894.txt |
| 86DD | USC/ISI 4676 Admiralty Way, Marina del Rey, CA  | IPv6 Internet Protocol Version 6 Transmission of Packets over Ethernet Networks, RFC-2464, Internet Society, Dec. 1998. http://www.ietf.org/rfc/rfc2464.txt |
| 888E  | IEEE 802.1 802.1 Chairc/o IEEE Piscataway, NJ  | IEEE Std 802.1X - Port-based network access control |
| 88B7 | IEEE 802.1 IEEE 802.1 Chair c/o IEEE Piscataway, NJ | 802 - OUI Extended Ethertype. This Ethertype value is available for public use and for prototype and vendor-specific protocol development, as defined in Amendment 802a to IEEE Std 802.  |
| 88F0 | IEEE P1451.0700 King Farm Blvd., Rockville, MD | IEEE P1451.0 Smart Transducer Interface for Sensors and Actuators http://grouper.ieee.org/groups/1451/0/private/ |
| A0ED | IETF 6lo working group c/o Internet Society, Reston, VA | When carried over layer 2 technologies such as Ethernet, this EtherType will be used to identify IPv6 datagrams using LoWPAN encapsulation as defined in IETF RFC 4944 Transmission of IPv6 Packets over IEEE 802.15.4 Networks |

#### Dispatch code

##### Multiplex ID field

The Multiplex ID field used in the MPX IE, is used to multiplex different upper-layer protocols. The Multiplex ID field takes one of two meanings, depending on its numeric value as follows:

1. If the value of this field is less than or equal to 1500, the Multiplex ID field takes values as specified in Table 20.
2. If the value of this field is greater than 1500, the Multiplex ID field indicates the Ethertype of the MAC client protocol.

|  |  |  |
| --- | --- | --- |
| Multiplex ID (decimal) | Multiplex ID (hex) | Description |
| 1 | 0x0001 | KMP |
| 2 | 0x0002 | WiSUN |
| 3–1279 | 0x0003–0x04ff | Reserved |
| 1381 | 0x0565 | Vendor specific, OUI extended |
| 1280–1380, 1382–1500 | 0x0500–0x0564, 0x0566–0x05dc | Reserved |

The Multiplex ID field is present if the Frame number is 0x00 and the Transfer type is 0b010, or if the Transfer type is 0b000. If the Transfer type is 0b001, the Multiplex ID is stored inside the Transaction ID field and the Multiplex ID field is omitted.

### PDE Primitives

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Request | Indication | Response | Confirm |
| PDE-DATA | X | X |  | X |
| PDE-CONFIG | X | X |  | X |
| PDE-PURGE | X |  |  | X |

## MMI Description

**Purpose**

* + Directs and may modify information from a protocol module SAP to the appropriate MAC SAP or another protocol module SAP
	+ Directs and may modify information from a MAC SAP to a protocol module SAP

**Overview**

* Provides multiplex and an 802.15.9 fragmentation service to the packets sent by the ULI protocol modules and sends them to either the MCPS-SAP, the MLME-SAP, or to another protocol module SAP. The process of sending the packets includes formatting the ULI IE i.e. prepending the appropriate headers into the payload of the frame for transmission.
* The interface between the MMI and the ULI function modules includes the Multiplex ID and the payload formatted as a payload IE to be sent or the payload received.
* The mechanism for the MMI, i.e. the ability to send the data to the proper SAP, will be similar to the mechanism defined in IEEE 802.15.9 for the multiplexed data service.

The MMI service consists of four primitives as shown in Table 1.

Table 1—Summary of MMI primitives

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Request | Indication | Response | Confirm |
| MMI-DATA | X | X |  | X |
| MMI-MGMT | X | X |  | X |
| MMI-CONFIG | X | X |  | X |
| MMI-PURGE | X |  |  | X |

The MMI data service delivers an MMI data payload from the protocol blocks to the MCPS-SAP after it packages them into a ULI IE or an MPX IE using the formats shown in Table 2 and Table 2a. The dispatch or EtherType ID indicates the ULI destination of the data payload.

Table 2

|  |  |
| --- | --- |
| Octets: 1 | Variable |
| ULI IE ID | Payload |

Table 2a

|  |  |  |
| --- | --- | --- |
| Octets: 1 | 2 | Variable |
| MPX IE ID | Dispatch/EtherType ID | Payload |

The formatted ULI IE or MPX IE is sent using the MCPS-DATA primitive via either Data or Multipurpose frames to the recipient device. At the recipient device, the ULI IE or MPX IE is delivered to the MCPS-SAP where the MMI data service delivers the data payload to the SAP of the protocol block or upper layer interface as identified by the dispatch/EtherType ID. Figure 2 illustrates this message sequence.

ULI

MAC This ULI IE is sent using the MCPS-DATA primitive via either Data or Multipurpose frames to the recipient device. At the recipient device, the ULI IE is delivered to the MCPS-SAP where the MMI data service delivers the data payload to the identified protocol block. Figure 2 illustrates this message sequence.

C

Protocol Bock

Protocol Block

MAC

ULI

MMI-DATA.request

MCPS-DATA.request

Data frame

MCPS-DATA.indication

ACK frame

MMI-DATA.indication

MCPS-DATA.confirm

MMI-DATA.confirm

The MMI management service takes an MMI management payload from the protocol blocks, packages it into a ULI IE or MPX IE as shown in Figure 1, delivers it to the MLME-SAP, and then using the MLME-IE-NOTIFY primitive it is sent via either Command or the Enhanced Ack frames to the recipient device. At the recipient device, the ULI IE or MPX IE is delivered to the MLME-SAP, where the MMI management service delivers the management payload to the identified ULI protocol block.

The MMI configuration service delivers an MMI configuration payload from the Management protocol block to the MLME-SAP or other protocol blocks. The configuration payload is formatted as per the appropriate IEEE 802.15.4 primitive accessed through the MLME-SAP.

The MMI-PURGE service provides a means to remove or abort pending transfers from the MMI transaction queue of the originator.

### MMI Data Service Primitives

# MMI-DATA.request

MMI-DATA.indication

MMI-DATA.confirm

The primitive parameters are described in Table 3.

# Table 3—MMI-DATA.request parameters

| Name | Type | Valid range | Description |
| --- | --- | --- | --- |
| SrcAddrMode | Enumeration | NONE, SHORT, EXTENDED | The source addressing mode for this MMI data. |
| DstAddrMode | Enumeration | NONE, SHORT, EXTENDED | The destination addressing mode for this MMI data. |
| DstPanId | Integer | 0x0000–0xffff | The PAN identifier of the entity to which the MMI data is being transferred. |
| DstAddr | — | As specified by the DstAddrMode parameter. | The address of the receiving (destination) device. |
| MultiplexId | Integer | 0x0000–0xffff | The higher-layer protocol using the MMI data service. ULI IE ID or MPX IE ID and the EtherType/Dispatch code |
| MmiData | Set of octets | — | The set of octets forming the IE data payload. |
| MmiHandle | Integer | 0x00–0xff | An identifier that can be used to refer to the particular primitive transaction; used to match a confirm primitive with the corresponding request. |
| SecurityLevel | Integer | 0–7 | The combination of Message Integrity Check and Encryption to be applied to the payload of the MMI data service. For encoding see Table 9-6 in IEEE Std 802.15.4. |
| KeyIdMode | Integer | As defined in Table 9-7 of IEEE Std 802.15.4. | The mode used to identify the key purportedly used by the originator of the received frame. This parameter is invalid if the SecurityLevel parameter is set to 0x00. |
| KeySource | Set of octets | As indicated by the KeyIdMode parameter. | The originator of the key purportedly used by the originator of the received frame. The KeySource field, when present, indicates the originator of a group key. If the Key Identifier Mode field indicates a 4-octet Key Source field, then the Key Source field shall be the macPanId of the originator of the group key right concatenated with the macShortAddress of the originator of the group key. If the Key Identifier Mode field indicates an 8 octet Key Source field, then the Key Source field shall be set to the macExtendedAddress of the originator of the group key. This parameter is invalid if the KeyIdMode parameter is invalid or set to 0x00 or set to 0x01. |
| KeyIndex | Integer | 0x01–0xff | The Key Index field allows unique identification of different keys with the same originator. It is the responsibility of each key originator to make sure that the actively used keys that it issues have distinct key indices and that the key indices are all different from 0x00. |
| Send-Multipurpose | Boolean | TRUE, FALSE | If TRUE, use the 802.15.4 Multipurpose frame type. If FALSE, use 802.15.4 Data frame type.  |

### MMI-DATA.confirm

# The MMI-DATA.confirm primitive reports the results of a request to transfer data to another device. The semantics of the MMI-DATA.confirm are as follows:

# MMI-DATA.confirm

# ( MmiHandle,MaxTransferSize, Status)

# The primitive parameters are described in Table 4. If there is no capacity to store the transaction, the Status will be set to TRANSACTION\_OVERFLOW. In case the other end aborts the transaction then the status will be set to TRANSACTION\_ABORTED and the MaxTransferSize is set to the value returned from the other end.

# Table 4—MPX-DATA.confirm parameters

| Name | Type | Valid range | Description |
| --- | --- | --- | --- |
| MmiHandle | Integer | 0x00–0xff | An identifier that can be used to refer to a particular primitive transaction; used to match a confirm primitive with the corresponding request. |
| MmiTransfer- Size | Integer | 0x0000–0xffff | In case of an aborted transaction this parameter can be returned from the other end to indicate the maximum size of transaction it can handle. In case another end did not give a maximum size, this is set to zero. |
| Status | Enumeration | SUCCESS,TRANSACTION\_OVERFLOW, TRANSACTION\_EXPIRED,CHANNEL\_ACCESS\_FAILURE, INVALID\_ADDRESS, NO\_ACK, COUNTER\_ERROR, FRAME\_TOO\_LONG, UNAVAILABLE\_KEY,UNSUPPORTED\_SECURITY, INVALID\_PARAMETER. TRANSACTION\_ABORTED | The status of the last MMI data transmission. |

### MMI-DATA.indication

# The MMI-DATA.indication primitive delivers a MMI payload from another device. The semantics of this primitive are as follows:

# MMI-DATA.indication

# ( SrcAddrMode, SrcPanId,SrcAddr, DstAddrMode, DstPanId, DstAddr, MultiplexId, MmiData, SecurityLevel, KeyIdMode, KeySource, KeyIndex)

# The primitive parameters are described in Table 5.

# Table 5—MMI-DATA.indication parameters

| Name | Type | Valid range | Description |
| --- | --- | --- | --- |
| SrcAddrMode | Enumeration | NONE, SHORT, EXTENDED | The source addressing mode for this MMI data payload. |
| SrcPanId | Integer | 0x0000–0xffff | The PAN identifier of the entity from which MMI data is being transferred. |
| SrcAddr | — | As specified by the SrcAddrMode parameter. | The address of the transmitting (source) device. |
| DstAddrMode | Enumeration | NONE, SHORT, EXTENDED | The destination addressing mode for this MMI data payload. |
| DstPanId | Integer | 0x0000–0xffff | The PAN identifier of the entity to which the MMI data is being transferred. |
| DstAddr | — | As specified by the DstAddrMode parameter. | The address of the receiving (destination) device. |
| MultiplexId | Integer | 0x0000–0xffff | The higher-layer protocol using the MMI data service. See 7.2.3 |
| MmiData | Set of octets | — | The set of octets forming the MPX data payload. |
| SecurityLevel | Integer | 0–7 | See Table 2. |
| KeyIdMode | Integer | 0x00–0x03 | See Table 2. |
| KeySource | Set of octets | As specified by the KeyIdMode parameter. | See Table 2. |
| KeyIndex | Integer | 0x01–0xff | See Table 2. |

### MMI PURGE Service Primitive(s)

MMI-PURGE.request

MMI-PURGE.confirm

### MMI-PURGE primitives

# The MMI-PURGE primitives provide a means to remove or abort pending transfers from the MMI transaction queue of the originator.

# The MMI-PURGE.request primitive allows the next higher layer to purge an MMI payload from the transaction queue.

# The semantics of the MPX-PURGE.request are as follows:

# MMI-PURGE.request

# (MmiHandle,SendAbort)

# The primitive parameters are described in Table 6.

# Table 6—MPX-PURGE.request parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Valid range | Description |
| MmiHandle | Integer | 0x00–0xff | An identifier that can be used to refer to a particular primitive transaction; used to match a MMI-PURGE.request primitive with the corresponding MMI-DATA.confirm primitive. |
| SendAbort | Boolean | TRUE, FALSE | If this parameter is TRUE and the transaction is still active, the MMI data service sends a MPX IE with an abort code to the other end indicating that the transaction was aborted. If this parameter is FALSE, the transaction is just purged locally, and no information is sent to the other end. |

# On receipt of the MMI-PURGE.request primitive, the MMI data service attempts to find in the transaction queue the payload indicated by the MmiHandle parameter. If a MMI payload has left the transaction queue, the handle will not be found, and the MMI payload can no longer be purged. If a MMI payload matching the given handle is found, the payload is discarded from the transaction queue, and optionally an abort message is sent to the other end, if the SendAbort parameter is TRUE. If an abort message is sent to the other end that will allow the other end to clear out its state immediately without waiting for the timeout.

# The MMI-PURGE.request will also issue a corresponding MCPS-PURGE.request to the MAC data service, provided it has an MCPS-DATA.request in process when the MMI-PURGE.request is called.

### MMI-PURGE.confirm

# The MMI-PURGE.confirm primitive allows the MMI data service to notify the next higher layer of the success of its request to purge a MMI payload from the transaction queue.

# The semantics of this primitive are as follows:

# MMI-PURGE.confirm

# (MmiHandle, Status)

# The primitive parameters are described in Table 7.

# Table 7—MMI-PURGE.confirm parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Valid range | Description |
| MmiHandle | Integer | 0x00–0xff | An identifier which can be used to refer to a particular primitive transaction; used to match a confirm primitive with the corresponding request. |
| Status | Enumeration | SUCCESS, INVALID\_HANDLE | The status of the request to purge MMI data from the transaction queue. |

### MMI Management Service Primitives

# MMI-MGMT.request

MMI-MGMT.indication

MMI-MGMT.confirm

MMI Configuration Service Primitives

MMI-CONFIG.request

MMI-CONFIG.indication

MMI-CONFIG.confirm

# MMI-DATA.request

# ( SrcAddrMode, SrcPanId,SrcAddr, DstAddrMode, DstPanId, DstAddr, MultiplexId, MpxData, SecurityLevel, KeyIdMode, KeySource, KeyIndex)

MCPS-DATA.request (

SrcAddrMode,

DstAddrMode,

DstPanId,

DstAddr,

Msdu,

MsduHandle,

HeaderIeList,

PayloadIeList,

HeaderIeIdList,

NestedIeSubIdList,

AckTx,

GtsTx,

IndirectTx,

SecurityLevel,

KeyIdMode,

KeySource,

KeyIndex,

UwbPrf,

Ranging,

UwbPreambleSymbolRepetitions,

DataRate,

LocationEnhancingInformationPostamble,

LocationEnhancingInformationPostambleLength,

PanIdSuppressed,

SeqNumSuppressed,

SendMultipurpose

FrakPolicy,

CriticalEventMessage

)

### ULI Profile Operation Primitives

* Type of operations
	+ ULM-CREATE-PROFILE()
	+ ULM-EXEC-PROFILE()
	+ ULM-GET-PROFILE()
	+ ULM-CHANGE-PROFILE()
	+ ULM-DELETE-PROFILE()

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Request** | **Indication** | **Response** | **Confirm** |
| ULM-CREATE-PROFILE | X |  | X |  |
| ULM-EXEC-PROFILE | X |  | X |  |
| ULM-GET-PROFILE | X |  | X |  |
| ULM-CHANGE-PROFILE | X |  | X |  |
| ULM-DELETE-PROFILE | X |  | X |  |

MPH-EXEC-PROFILE.request

Management

Protocols sublayer

PDE

sublayer

Next

higher layer

**ULM-EXEC-PROFILE.request**

MMI-MGMT.request

MAC PIB

Device

MAC

MLME-SRM.request

MMI

sublayer

MPH-EXEC-PROFILE.response

**ULM-EXEC-PROFILE.response**

MMI-MGMT.response

MLME-SRM.response

IEEE802.15.12

MLME-SET.request

MLME-SET.response

IEEE802.15.4

### ULI Protocol Module Configuration Primitives

* Type of operations
	+ ULM-LIST-MODULES(): retrieve supported protocol module(s)
	+ ULM-GET-MODULE-STATUS(): get the status of the protocol module
	+ ULM-SET-MODULE-STATUS(): set the status of the protocol module

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Request** | **Indication** | **Response** | **Confirm** |
| ULM-LIST-MODULES | X |  | X |  |
| ULM-GET-MODULE-STATUS | X |  | X |  |
| ULM-SET-MODULE-STATUS | X |  | X |  |

### Usage of ULM-LIST-MODULES()

The entire list of module attributes are collected in a YANG store called “MODULE-DESCRIPTOR"

“ULM-LIST-MODULES” ULI() is an enumeration function:

 **handle = ULM-LIST-MODULES( NULL, module\_descriptor\_result)**

the first module descriptor returned is in “module\_descriptor\_result”

the first module returned is always the “Management Module”

Then the caller makes additional calls as follows:

 **handle = ULM-LIST-MODULES(handle, module\_descriptor\_result)**

Each time returning  a module descriptor in “module\_descriptor\_result”

The caller can know it reached the end of the module list when “handle” returns NULL

Management Protocol

KMP

L2R

Ranging

NULL

**PTM Description**

**Design of (and questions about) the ULI Passthru Module**

The purpose of this document is to identify possibilities for PTH-SAP and PTM-SAP – i.e., the interfaces of the mandatory Passthru functional block of the 802.15.12 ULI architecture as shown in Figure:

From one point of view, there should be very little specification required for such a module, since it does not seem to impose any process requirements on the data that it passes through to the next lower layer (for transmission) or next higher layer (for reception). Nevertheless, there are several possibilities which make sense. Do we provide for all of them? If only one, how do we choose?

* For outgoing data, the Passthru module could require that the higher layer identify all 28 parameters for presentation of the payload (via PTM-SAP) to the MCPS-SAP.
* For control operations, the Passthru module could require that the higher layer identify the specific primitive desired, as well as all parameters of each primitive
* For outgoing data, the Passthru module could emulate the operation of IP raw sockets. In this mode, the higher layer would pass a completely (or nearly completely) formed frame to the lower layer. This might be useful for diagnostics, but seems unrelated to the specific goals of ULI.
* In order to reduce the number of parameters required by the Passthru module, it could be allowed for the higher layer protocol to pass a profile identifier as one of the parameters. Then, most of the parameters would be implicitly available to the Passthru module by referring to the definition and current configuration state for that profile within the ULI layer.

Upon reception of a well-formed 802.15.4 frame, the Passthru module should be able to dispatch the frame by inspection of the frame fields following the MHR. But this is what the ULI must anyway do upon frame reception, so that it is not clear whether the Passthru module actually has any responsibility for handling incoming frames.

* Does the Passthru module bypass L2R routing operations? If so, then transmission could occur only to immediate neighbors. Similarly, if ranging is required prior to transmission, how does Passthru know to make the request?
* If profiles are not enabled for use with the Passthru function, how does the function identify the proper MAC interface? Are the appropriate MAC and PHY layers associated with the source MAC address of the frame? Or, with the destination MAC address?

If profiles are enabled for use, then a table of profiles will be required, along with the MCPS-SAP interface parameters that each profile allows the Passthru module to infer from the profile definition. In this way, most of the Passthru specification text actually would reside as part of the profile definitions. Each new profile will need a section within its specification for use by the generic Passthru module.

For reference during discussion of the above questions, here is a representation of the parameters required for transmitting a frame by way of ULI:

**802.15.4**

MCPS-DATA.request (

SrcAddrMode,

DstAddrMode,

DstPanId,

**DstAddr,**

**Msdu**,

MsduHandle,

HeaderIeList,

PayloadIeList,

HeaderIeIdList,

NestedIeSubIdList,

AckTx,

GtsTx,

IndirectTx,

SecurityLevel,

KeyIdMode,

KeySource,

KeyIndex,

UwbPrf,

Ranging,

UwbPreambleSymbolRepetitions,

DataRate,

LocationEnhancingInformationPostamble,

LocationEnhancingInformationPostambleLength,

PanIdSuppressed,

SeqNumSuppressed,

SendMultipurpose

FrakPolicy,

CriticalEventMessage

)

Some parameters aren’t simple data types, and can have substructure and refer to objects with nontrivial structure. We will need NULL (*not present*) parameters.