



The MailBox Profile for ZigBee

Multipoint asynchronous serial communications profile for ZigBee

Summary

The MailBox profile is a ZigBee Application Sublayer for multipoint asynchronous data communication. It provides serial communications in the sense that, unless an error is asserted, data frames are guaranteed to arrive in sequence. This lends itself readily to serial cable/bus replacement applications.

In addition, the Mailbox profile provides generic communications for applications without a dedicated ZigBee profile, for example:

- application wishing to take advantage of existing MailBox devices such as serial and USB adapters
- where communications are to be bridged to non-ZigBee media such as TCP/IP or RS485
- where the market size does not merit development of a dedicated ZigBee profile

A network-wide sleep feature is also provided which allows a network or parts of the network to agree to allow all devices to sleep until a later wakeup time.

The profile is public. FlexiPanel Ltd retains the right to define the profile, but attaches no limitations as to its adoption and use. If desired, FlexiPanel can certify an implementation as compliant.

Features

MailBox profile incorporates the following features:

- conveyance of arbitrary data payloads in sequence at up to approx 19.2kbit/s
- addressed or broadcast (bus) messaging
- function-specific functional address system independent of underlying ZigBee addresses
- payload acknowledgement and failure management
- network-wide sleep and wakeup
- sleepy end device sleep & wakeup

- custom messages, e.g. for modem status signals and application reflashing

MailBox devices can share the network with non-MailBox devices. However, non-MailBox devices must correctly ignore broadcast messages generated with the MailBox application profile ID.

Stack Profile

The 1.0 implementation of the MailBox profile is based on version 1.0 of the ZigBee specification.

The MailBox protocol may be implemented using any ZigBee Stack Profile, and therefore MailBox devices can piggyback on any existing ZigBee network. However, of five principle profiles are proposed, depending on the application and its environment:

HC-MailBox: Uses the Home Controls stack profile in order to be compatible with existing Home Controls ZigBee networks. (Refer to ZigBee Specification 1.0, section E3 for profile details.)

BA-MailBox: Uses the Building Automation stack profile in order to be compatible with existing Building Automation ZigBee networks. (Refer to ZigBee Specification 1.0, section E4 for profile details.)

PC-MailBox: Uses the Plant Control stack profile in order to be compatible with existing Building Automation ZigBee networks. (Refer to ZigBee Specification 1.0, section E5 for profile details.)

R2-MailBox: Uses the Route-2 stack profile defined in Table 1. Each node may have two routers as children. This provides a network radius of 15 hops to the coordinator, and enables cable replacement systems with considerable multidimensional span without seriously undermining network latency.

The Route-2 Profile is intended for extended networks. With an inter-node distance of 100m, a network could span 3km while taking no more than a second to send a message from end to end.

Table 1. Route-2 Stack Profile	
Stack Profile ID	0x00
nwkProtocolVersion	0x01
nwkSecurityLevel	0x00 / 0x05
nwkMaxChildren	0x02
nwkMaxDepth	0x0F
nwkMaxRouters	0x02
MinBindings	100
MinCoordinatorNeighbors	24
MinRouterNeighbors	25
MinRouteDiscoveryTableSize	4
MinRoutingTableSize	8
MinReservedRoutingTableEntries	8

LN-MailBox: Uses the *Linear* stack profile defined in Table 2. Each node may have one router as a child. This provides a large network “radius”; however, the coordinator must be at one end of the network and the remaining nodes must extend linearly from it.

The ZigBee specification uses a one-byte network depth parameter in several places. This limits the maximum length of Linear Profile to 256 elements. It will be proposed to the ZigBee Alliance that if the depth parameter equals 0xFF, this will imply that the network depth is instead inferred from the device short address. This increases the length to 65536 nodes.

Table 2. Linear Stack Profile	
Stack Profile ID	0x00
nwkProtocolVersion	0x01
nwkSecurityLevel	0x00 / 0x05
nwkMaxChildren	0x01
nwkMaxDepth	0xFFFF
nwkMaxRouters	0x01
MinBindings	100
MinCoordinatorNeighbors	24
MinRouterNeighbors	25
MinRouteDiscoveryTableSize	4
MinRoutingTableSize	8
MinReservedRoutingTableEntries	8

Both the Route-2 and Linear profiles employ sparse tree connectivity. This does not necessarily undermine network robustness, since routing tables will still provide the opportunity to bypass faulty nodes.

Device & Profile IDs

The MailBox has been use the following IDs which have been accepted by the ZigBee Alliance for acceptance.

Profile ID	0xC1EE
MailBox Device ID	0xF000
Device ID Version #	0x00

Notation, Byte & Bit order

In this profile definition, all numbers quoted are in decimal unless prefixed with 0x, in which case they are hexadecimal. Index counting starts at zero, so the first byte of a message is byte zero.

Multi-byte data is transmitted least-significant byte first ('little-endian'), as is standard in the ZigBee specification.

In cases where multiple fields are packed into a single byte, those fields shall be signified using [] to indicate the byte and then a colon to indicate the bit-fields. For example bits 4 to 6 of byte 3 are denoted by [Byte 3]:4-6.

Functional Clusters

The MailBox devices use clusters to identify themselves according to the functions they perform. All devices support cluster 0x0000 and they may support any number of other clusters. To do so, they must:

- respond appropriately to *Match_Desc_req* requests
- only process messages for which they provide cluster support

All clusters process data in an identical manner. The purpose of using clusters in this way is:

- to provide function-specific device and service discovery
- to provide a method for transmitting to a subset of MailBox devices

The interpretation of cluster values 0x0001-0x00FD is application specific and not defined in the protocol. For example, in a sensor network, all sensors might support clusters 0x0000 and 0x0001. Data gathering gateways might support clusters 0x0000 and 0x0002. A gateway could send a message to all sensors by broadcasting to cluster 0x0001. A sensor can search for a data gateway using the *Match_Desc_req* service and then direct MailBox packets directly to it. (It would probably not broadcast to cluster 0x0002 unless a large number of gateways existed.)

It is anticipated that 2-byte clusters will be supported in ZigBee 1.1. At that time, the range of application specific clusters will be extended to 0x0001-0xFFFFD.

Cluster 0xFFFFE shall be the Redirect address signifier. When 0xFFFFE is specified as a destination address, the destination specified by the most recently received Redirect Address message. No device shall support cluster 0xFFFFE directly.

Cluster 0xFFFF is used to signify a null cluster which may be used as padding in fixed-length cluster lists. No device shall support cluster 0xFFFF.

Application ID

The use of functional clusters alone does not allow for the fact that two applications may need to coexist on a network. Specifically, one manufacturer may associate different meanings to different clusters.

To allow for this, a 4-byte application ID is associated with each manufacturer's interpretation of clusters. This application ID is included with all data transactions and is also available for direct querying to verify a device prior to unicast transmission. It also allows manufacturer specific messages to be transmitted.

In order for application IDs to be unique, FlexiPanel will allocate a 3-byte MailBox Unique Identifier (MUI) on request. At present this will be free of charge.

Application IDs shall be allocated as follows:

Bytes 0-2: Manufacturer's MUI number
Byte 3: Assigned by manufacturer

The 'free-for-all' MUI 00:00:00 may be used by any application that can ensure that it is the only MailBox application on the network. Other than the 'Free-For-All' application ID, no MUI should be employed without the permission of the MUI owner.

Endpoints

The MailBox protocol uses one endpoint. This may be any endpoint in the allowed range 1-240.

The same endpoint must be used for all devices employing that Application ID.

Sequencing & Acknowledgement

Two data transfer options are provided:

- *Acknowledged.* The destination must accept a frame before the source transmits the next frame. Broadcast transmissions are not permitted at present but will be in future.
- *Unacknowledged.* Frame sequence numbers are used to determine whether a frame arrives out of sequence or is lost. Loss of sequence is reported to the receiving application but no other action is taken. No acknowledge is sent to the source. Broadcast communication is permitted.

Power Outage Recovery

It should be expected that power outages will occur. In order to provide automatic recovery from power outages:

- All devices (except the coordinator) should reset if they fail to communicate with their parent. Such a failure will be indicated by a stack failure status code or a Sync-Loss.indication.
- It is recommended that mains-powered devices set *PollRate* MIB attribute to non-zero in order to regularly test contact with the parent.
- Upon reset, all devices should automatically perform a Join.request in order to rejoin the network. If rejoining fails, the device should continue to retry. If it is battery powered, it may opt to sleep for a suitable period between retries.

MailBox Frame Format

All MailBox transactions use MSG data transfer format, as opposed to the KVP transfer format, and consist of one MSG transaction only.

All frames have the format shown in Table 3.

Table 3. MailBox MSG format	
Description	Bytes
Frame ID	1
Frame payload	Implied by ZigBee packet size

Valid frame identifiers (FIDs) are listed in Table 4.

Table 4. MailBox frame identifiers	
Message	FID
Application ID request	0x01
Application ID response	0x02
Data	0x03
Acknowledge	0x04
Network Sleep	0x05
Present	0x06
Redirect Address	0x07
Manufacturer specific frames	0x80-0xFF

The frames are interpreted exactly the same way irrespective of the Cluster ID.

Application ID request

Any device may request the 4-byte Application ID of any other device by issuing an *Application ID request* frame.

Application ID request frames have the format shown in Table 5.

Table 5. Application ID request format	
Description	Byte fields
Application ID request FID (0x01)	Byte 0

Application ID response

A device responds to an *Application ID request* by generating an *Application ID response* frame.

Application ID response frames have the format shown in Table 6.

Table 6. Application ID response format	
Description	Byte fields
Application ID response FID (0x02)	Byte 0
Application ID	Bytes 1-4

If an unexpected *Application ID response* frame is received, it shall be ignored.

Data

A device sends data to other applications by issuing a *Data* frame. A maximum of 64-bytes of data may be transferred.

Data frames have the format shown in Table 7.

Table 7. Data format	
Description	Byte fields
Data FID 0x03	Byte 0
Application ID	Bytes 1-4
Sequence Number	Byte 5
Broadcast	[Byte 6]:0
Acknowledged	[Byte 6]:1
Reserved	[Byte 6]: 2-7
Data payload	Up to 64 bytes
Checksum	Last byte

The *Broadcast* flag specifies than the transmission was broadcast.

The *Acknowledged* flag specifies than an Acknowledge frame should be generated on receipt of the Data frame.

The *Sequence Number* is a value provided to test whether frames are arriving in sequence.

The length of the *Data payload* shall be implied by the ZigBee frame length.

The *Checksum* shall be such that the non-carrying sum of all bytes listed in Table 7 is 0x00.

Acknowledge

The *Ack* frames have the format shown in Table 8.

Table 8. Acknowledge format	
Description	Byte fields
Acknowledge FID 0x04	Byte 0
Acknowledge type: 0x00 = Success 0x01 = Unknown 0x02 = Sequence Error 0x03 = Not permitted 0x05 = Retry Later 0x09 = Reset mismatch 0x0A = Checksum failure	Byte 1
Sequence number of original transmission	Byte 2
RetryDelay	Bytes 3-4

The *Acknowledge type* indicates success or reason for failure. If the reason for failure is *Checksum failure*, the application may elect to reattempt transmission immediately. If the reason for failure is *Retry Later*, the application may elect to reattempt transmission should occur no sooner than $256 \times \text{RetryDelay}$ symbol periods later. In neither case does the MailBox layer does not automatically retry. The retried transmission shall use the same sequence number as the original transmission.

To ensure an acknowledge frame is valid, the sequence number and source address shall be verified. If an invalid or unexpected acknowledge frame is received, it shall be ignored.

Network Sleep

A device may instruct other devices to sleep using the *Network Sleep* frame. The recipient devices may then enter a sleep state for the specified duration.

No device shall sleep if operation if any non-MailBox functions in the ZigBee network would be affected. It is up to the application to determine whether this would be the case.

Network Sleep frames have the format shown in Table 9.

Table 9. Network Sleep format	
Description	Byte fields
Network Sleep FID 0x05	Byte 0
Application ID	Bytes 1-4
Closedown period	Bytes 5-6
Sleep period	Bytes 7-9
Wakeup period	Bytes 10-11

The *Closedown period* is the duration, in seconds, that the device should continue to keep its radio on and respond to messages if required.

The *Sleep period* is the duration, in seconds, that the device may sleep.

The *Wakeup period* is the duration, in seconds, immediately following the end of the *Sleep period*. During this period, the device shall not initiate any transmissions unless it first receives a transmission from another device.

Present

A device may announce its presence to other devices using the *Present* frame. Typically this would occur as a broadcast after joining or rejoining a network.

The Present frame has the format shown in Table 10.

Table 10. Present format	
Description	Byte fields
Present FID 0x06	Byte 0
Application ID	Bytes 1-4
Number of Functional Clusters	Byte 5
Functional Cluster List	2 bytes each

The *Function Cluster List* is a list of clusters supported by the application. The mandatory cluster 0x0000 is not required to be included in the list.

The *Number of Functional Clusters* is the number of clusters in the list.

Redirect Address

A device may instruct other devices where to send messages using the *Redirect Address* frame.

The Redirect Address frame has the format shown in Table 11.

Table 11. Redirect Address format	
Description	Byte fields
Present FID 0x07	Byte 0
Application ID	Bytes 1-4
<i>IsBroadcast</i>	[Byte 5]:0
<i>IsAcknowledge</i>	[Byte 5]:0
Reserved	[Byte 5]: 1-7
Redirect Address	Bytes 6-7

If *IsBroadcast* is true, *Redirect Address* shall be interpreted as a functional cluster to broadcast to. If *IsBroadcast* is false, *Redirect Address* shall be interpreted as a short address to unicast to.

Whenever a device uses the Redirect Cluster (0xFFFE) for data transmission, the *Redirect Address* of the most recently received Redirect Address message shall be the actual destination to which the data is dispatched. If *IsAcknowledge* is true, the transmission shall be acknowledged. If no such Redirect Address has been specified, the destination shall be *MIB_DefaultRedirectAddress*.

Custom

FIDs 0x80 to 0xFF are manufacturer-specific frames provided to allow manufacturers to provide auxiliary functionality over the MailBox layer.

These frames are intended for features such as reflashing the application-layer over the network, and for providing Modem Status indications. No limitation is set as to their use.

Application frames have the format shown in Table 12.

Table 12. Custom frame format	
Description	Byte fields
Customer FID (0x80 – 0xFF)	Byte 0
Application ID	Bytes 1-4
Additional manufacturer specific data	As required

MailBox Functional Description

Sequence number (SN) notation

Devices shall maintain a *Sequence Buffer* of records of the format depicted in Table 13:

Table 13. Sequence buffer record

Sequence number <i>Q</i> or <i>P</i>	1 byte
True if Broadcast	1 bit
True if record is a <i>P</i> SN, false if <i>Q</i>	1 bit
Destination address or cluster (if <i>Q</i>)	2 bytes
Source address (if <i>P</i>)	2 bytes
Source cluster (if <i>P</i> and Broadcast)	2 bytes

The following notation is used:

S is the sequence number of current frame about to be transmitted or just received.

Q is the sequence number of that source's last frame transmitted to that Destination or Cluster.

Q values are stored in the Sequence buffer as they are transmitted. If an entry already exists for the Address (if unicast) or Cluster (if broadcast), the earlier entry is discarded. If the buffer is full, the oldest entry is discarded. (No differentiation is made with respect to acknowledged and unacknowledged transmissions, the same sequence record applies to both.)

If no *Q* value exists for an Address or Cluster and the buffer is not full, *Q* shall be 0xFF. This shall imply this is the first such value since device reset.

If no *Q* value exists for as Address or Cluster and the buffer is full, *Q* shall be 0xFE. This shall imply that the previous *Q* value could not be determined.

P is the sequence number of the last frame received from that Source-Cluster pair which was not discarded by the destination.

P values are stored in the Sequence Buffer as they are transmitted. If an entry already exists for the Source-Cluster pair, the earlier entry is discarded. If the buffer is full, the oldest entry is discarded.

x+ shall indicate the next number in the sequence, where *x* = *Q* or *P*. It is defined as:

$x+ = x+1$	For $x < 0xFD$
$x+ = 0x00$	For $x \geq 0xFD$
$x+ = 0xFF$	If x does not exist and the table is not full.
$x+ = 0xFE$	If x does not exist and the table is full.

S shall be called a *late frame* if:

$S = 0xFF$ and $P < 0xFE$, or

$(S - P) \bmod 0x100 > 0x80$ and $S, P < 0xFE$

Join.request

MailBox devices will initialize and join a ZigBee network in the same manner as any other ZigBee device. No other MailBox task may be performed succeed until the device is successfully joined to a network.

The application may attempt to join a ZigBee network by sending a *Join.request* primitive to the MailBox layer with the parameters shown in Table 14.

Table 14. <i>Join.request</i> fields	
Description	Size
Erase flag	1 bit

Upon receipt of the *Join.request* primitive, the MailBox layer shall:

- Perform any stack initialization required.
- If the *Erase* flag is true, any network, binding and routing tables are erased and the device shall assume it is not a member of a network.
- Coordinators will form a network using the NMLE-Network-Formation.request primitive.
- Routers will join a network using the NMLE-Join.request primitive. If the join is successful, the router will begin routing using the NMLE-Start-Router.request.
- End devices will join a network using the NMLE-Join.request primitive.
- When using the NMLE-Join.request primitive, devices which are already members of a network shall join with the *RejoinNetwork* field set to *true*. Otherwise it shall be *false*. An already-joined device can only join a new network if the *Erase* parameter is *true*

Join.confirm

Upon completion of the operation, the MailBox layer shall return a *Join.confirm* primitive to the application layer with the parameters shown in Table 15.

Table 15. <i>Join.confirm</i> fields	
Description	Size
Status	1 byte

The Status shall be the status of the last NWK layer confirmation received.

Permit-Join.request

If the device is a coordinator or router, the application may instruct the MailBox layer whether or not to allow other ZigBee nodes to join by issuing a *Permit-Join.request* with the parameters shown in Table 16.

Table 16. <i>Permit-Join.request</i> fields	
Description	Size
Permit Duration	1 byte

Permit Duration is the duration, in seconds, for which joining should be permitted. If 0x00, joining shall not be permitted. If 0xFF, joining shall be permitted indefinitely. Upon receipt of the *Permit-Join.request* primitive MailBox will issue a NMLE-Join.request primitive to the NWK layer.

Permit-Join.indication

If this device has permitted another device to join the network, the MailBox layer shall return a *Permit-Join.indication* primitive to the application layer with the parameters shown in Table 17.

Table 17. <i>Permit-Join.indication</i> fields	
Description	Size
Short address of joining device	2 bytes
Long address of joining device	8 bytes

Permit-Join.confirm

Upon completion of the operation, the MailBox layer shall return a *Permit-Join.confirm* primitive to the application layer with the parameters shown in Table 17.

Table 18. <i>Permit-Join.confirm</i> fields	
Description	Size
Status	1 byte

The Status shall be the Status of the NMLE-Permit-Join.request.

Leave.indication

A *Leave.indication* primitive is generated if this device is instructed to leave the network. This is provided for compatibility with non-MailBox nodes only.

Sync-Loss.indication

A *Sync-Loss.indication* indicates this device did not receive a reply from its parent during a poll operation. Polling will have been tried four times; the application should assume that the parent is no longer operating, on this frequency at least. The recommended course of action is to sleep a while to conserve power, then reset and then attempting to rejoin the network.

Device-Discovery.request

The application may discover other MailBox devices on the ZigBee network with by issuing a *Device-Discovery.request* containing the fields shown in Table 19.

Table 19. <i>Device-Discovery.request</i> fields	
Description	Size
Functional Cluster ID	2 bytes

Upon receipt of the *Device-Discovery.request* primitive, the MailBox layer shall broadcast a *Match_Desc_req* to all devices on the network, specifying the Functional Cluster ID as both an input and output Cluster.

Device-Discovery.indication

If a *Device-Discovery.confirm* has not yet been generated in response to a *Device-Discovery.confirm*, a *Device-Discovery.indication* primitive shall be generated for every device discovered in *Match_Desc_rsp* responses. The indication shall contain the fields shown in Table 20.

Table 20. <i>Device-Discovery.indication</i> fields	
Description	Size
Discovered device short address	2 bytes

Device-Discovery.confirm

After two *MailBoxTimeout* periods have occurred to ensure that no further responses are due, the MailBox layer shall return a *Device-Discovery.confirm* primitive to the application layer with the parameters shown in Table 21.

Table 21. <i>Device-Discovery.confirm</i> fields	
Description	Size
Status	1 byte

The *Status* shall zero if the operation completed without error. If any *Match_Desc_rsp* responses are thereafter received, they shall be ignored.

Service-Discovery.request

The application may request information about any other device on the ZigBee network with by issuing a *Device-Discovery.request* containing the fields shown in Table 22.

Table 22. <i>Service-Discovery.request</i> fields	
Description	Size
Device short address	2 bytes
Information type ≤0x007F = ZDO request cluster 0x0080 = Mailbox Application ID	2 bytes
ZDO request payload	As required

Upon receipt of the *Device-Discovery.request* primitive, if the information type is MailBox Application ID, the MailBox layer shall send an Application ID request frame to the other device. Otherwise, it shall send a ZDO request to the cluster identified, using the arguments provided in the ZDO request payload. (Note: *Device-Discovery* should be used for *Match_Desc* ZDO requests, rather than *Service-Discovery*.)

Service-Discovery.confirm

Upon receipt of an Application ID response frame, a *Service-Discovery.confirm* response shall be generated containing the 4-byte Application ID in the result payload and Status of 0x00. If no response to an Application ID request frame is received within a period of MailBoxTimeout, a *Service-Discovery.confirm* response shall be generated with Status 0xEB (NO_DATA). Refer to Table 23

Table 23. <i>Service-Discovery.confirm</i> fields (App ID)	
Description	Size
Status	1 byte
Remaining ZDO payload	As required

Upon receipt of a ZDO confirmation, a *Service-Discovery.confirm* response shall be generated containing the ZDO response payload. The first byte shall be interpreted as the confirm Status byte as shown in Table 24.

Table 24. <i>Service-Discovery.confirm</i> fields (ZDO)	
Description	Size
Status	1 byte
Remaining ZDO payload	As required

The *Status* shall zero if the operation completed without error. If any ZDO confirmation is received unexpectedly, it shall be ignored.

Data.request

The source application initiates the transfer of a packet of data by issuing a *Data.request* containing the fields in Table 25.

Table 25. <i>Data.request</i> fields	
Description	Size
True if <i>Broadcast</i>	1 bit
True if <i>Acknowledged</i>	1 bit
True if <i>Retry</i>	1 bit
Destination	2 bytes
Data payload	≤64 bytes

If *Destination* represents a Functional Cluster to broadcast to, *Broadcast* shall be true. If *Destination* represents a short address to unicast to, *Broadcast* shall be false.

If *Acknowledge* is true, confirmation will only be returned once receipt of the packet is acknowledged. Broadcast transmissions may not be acknowledged.

Retry should only be true if this is a repeat attempt to retransmit a packet which was rejected by the recipient due to an acknowledge status of *Retry Later*, *Timed Out* or *Checksum Failure*. If retry is to be attempted, it must be the *Data.request* immediately following the failed transmission. Retry is not required; transmission of the data may be abandoned by the application.

Upon receipt of a *Data.request* the MailBox layer will generate a Sequence number *S* as follows.

- If a corresponding *Q* value exists in the *Q* Buffer and *Retry* is false, $S = Q+$.
- If a corresponding *Q* value exists in the *Q* Buffer and *Retry* is true, $S = Q$.
- If no corresponding *Q* value exists and the *Q* Buffer is not full, *S* shall be 0xFF.
- If no corresponding *Q* value exists and the *Q* Buffer is full, *S* shall be 0xFE.

The MailBox layer will then transmit a Data frame according to the Acknowledged Transfer Sequence or Unacknowledged Transfer Sequence as described below.

Data.confirm

The Status of the transfer is conveyed back to the application with a *Data.confirm* containing the fields in Table 26:

Table 26. <i>Data.confirm</i> fields	
Description	Size
Status 0x00 = Success 0x01 = Unknown 0x02 = Sequence Error 0x03 = Not permitted 0x04 = Timed out 0x05 = Retry Later 0x06 = Lower-level stack failure 0x09 = Reset mismatch 0x0A = Checksum failure	1 byte
Sequence number <i>S</i> , or ZigBee status value if stack failure	1 byte
RetryDelay if Retry Later	Bytes 3-4

A *Lower-level stack failure* is reported if the MailBox layer cannot complete its task because of an error at a lower layer in the stack.

Data.indication

On receipt of a *Data* frame, the destination MailBox layer reports the frame to its application layer by generating a *Data.indication* primitive containing the fields in Table 27.

Table 27. <i>Data.indication</i> fields	
Description	Size
Status 0x00 = Success 0x01 = Unknown 0x02 = Sequence Error 0x06 = Stack failure 0x07 = Frames lost 0x08 = Late frame 0x09 = Reset mismatch	1 byte
Broadcast	1 bit
Acknowledged	1 bit
Sequence number <i>S</i> , or ZigBee status value if stack failure	1 byte
Source short address	2 bytes
Broadcast Cluster, if <i>Broadcast</i>	2 bytes
Data payload	≤64 bytes

Acknowledged Transfer Sequence

Upon receipt of a *Data.request* with Acknowledged equal to true, and *Broadcast* set to false, the source MailBox layer will transmit a data frame to the destination using the *Data* frame format. The Functional Cluster used shall be 0x0000.

When the destination receives a *Data* frame:

- If the Application ID is not correct, nothing is reported to the application layer. If required, an *Acknowledge* frame will be sent to the source with *Acknowledge Type* equal to *Not permitted*.

Upon receipt of the *Acknowledge* frame, the source MailBox layer shall issue a *Data.confirm* with Status value *Not permitted*.

- Otherwise, if the *RetryDelay* MIB value is non-zero, the frame will be discarded. An *Acknowledge* frame will be sent to the source with *Acknowledge Type* equal to *RetryLater*.

If an *Acknowledge* is received with *Acknowledge type* equal to *Retry Later*, the application may retry transmission provided a total period of *RetryTimeout* has not elapsed since the original *Data.request*. The MailBox layer does not automatically retry. The retried transmission shall not use the same sequence number as the original transmission.

- Otherwise, if an incorrect checksum does not permit acceptance of the frame, it will be discarded. An *Acknowledge* frame will be sent to the source with *Acknowledge Type* equal to *Checksum Failure*.

If an *Acknowledge* is received with *Acknowledge type* equal to *Checksum Failure*, retransmission may be reattempted immediately a maximum *MaxReTx* times before issuing a *Data.confirm* with Status value *Timed Out*.

- Otherwise, if *P* exists and *S* = *P*, it shall discard the frame. If required, an *Acknowledge* frame will be sent to the source with *Acknowledge Type* equal to *Success*.

Upon receipt of the *Acknowledge* frame, the source MailBox layer shall issue a *Data.confirm* with Status value *Success*.

- Otherwise, if *S* = 0xFE or *P+* = 0xFE, it will report the packet to the Application layer using a *Data.indication*. The Status value shall be *Unknown*. If required, an *Acknowledge* frame will be sent to the source with *Acknowledge Type* equal to *Unknown*.

Upon receipt of the *Acknowledge* frame, the source MailBox layer shall issue a *Data.confirm* with Status value *Unknown*.

- Otherwise, if *S* = 0xFF or *P+* = 0xFF but not both, it will report the packet to the Application layer using a *Data.indication*. The Status value shall be *Reset Mismatch*. If required, an *Acknowledge* frame will be sent to the source with *Acknowledge Type* equal to *Reset Mismatch*.

Upon receipt of the *Acknowledge* frame, the source MailBox layer shall issue a

Data.confirm with Status value *Reset Mismatch*.

- If $S = P+$, the frame is accepted. It will report the packet to the Application layer using a *Data.indication*. If required, an *Acknowledge* frame will be sent to the source with *Acknowledge Type* equal to *Success*.

Upon receipt of the *Acknowledge* frame, the source MailBox layer shall issue a *Data.confirm* with Status value *Success*.

- Otherwise, an error has occurred. It will report the error to the Application layer using a *Data.indication* with a Status value of *Sequence Error*. If required, an *Acknowledge* frame will be sent to the source with *Acknowledge Type* equal to *Sequence Error*.

Upon receipt of the *Acknowledge* frame, the source MailBox layer shall issue a *Data.confirm* with Status value *Sequence Error*.

- If the source does not receive an *Acknowledge* frame within a period of *MailBoxTimeout* since transmission, a *Data.confirm* is issued with Status value *Timed Out*.

Upon receipt of a *Data.request* with Acknowledged equal to true, and *Broadcast* set to true, a *Data.confirm* will immediately be issued with Success value equal to *Not permitted*.

Unacknowledged Transfer Sequence

Upon receipt of a *Data.request* with *Sequencing* set to *RMF*, the source MailBox layer will transmit a data frame to the destination(s) using the *Data* frame format. If unicast, the Functional Cluster used shall be 0x0000. It shall then issue a *Data.confirm* with Status value equal to *Success*.

Upon receipt of the Data frame:

- If the Application ID is not correct, nothing is reported to the application layer.
- Otherwise, if the *RetryDelay* MIB value does not permit acceptance of the frame, it will be discarded.
- Otherwise, if an incorrect checksum does not permit acceptance of the frame, it will be discarded.
- Otherwise, if P exists and $S = P$, it shall discard the frame.

- Otherwise, if $S = 0xFE$ or $P+ = 0xFE$, it will report the packet to the Application layer using a *Data.indication*. The Status value shall be *Unknown*.
- Otherwise, if $S = 0xFF$ or $P+ = 0xFF$ but not both, it will report the packet to the Application layer using a *Data.indication*. The Status value shall be *Reset Mismatch*.
- Otherwise, if $S = P+$, the frame is accepted. It will report the packet to the Application layer using a *Data.indication*.
- Otherwise if $S \neq P$ and $S \neq P+$, it will report the packet to the Application layer using a *Data.indication*. If S is determined to be a *late frame*, the Status value shall be *Late Frame*, otherwise it shall be *Frames Lost*.

Device-Sleep.request

The *Sleep.request* applies only to end devices that are permitted to sleep for short periods, i.e. *Rx off when idle* is true. While sleeping, a device's parent must store messages for it.

The application may instruct the device that it wishes to sleep by issuing a *Device-Sleep.request*.

Device-Sleep.confirm

When the MailBox has prepared for sleeping, it shall return a *Device-Sleep.confirm* primitive to the application layer with the parameters shown in Table 28.

Table 28. <i>Device-Sleep.confirm</i> fields	
Description	Size
Status	1 byte

The Status shall be 0x00 if the device may sleep.

Device-Wake.request

The *Device-Wake.request* applies only to end devices where *Device-Sleep.request* applies.

The application may instruct the device that it has just woken from single-device sleep by issuing a *Device-Wake.request*.

Device-Wake.confirm

Upon receipt of the *Device-Wake.request*, the MailBox layer shall resume stack operation and return a *Device-Wake.confirm* to the application layer with the parameters shown in Table 29.

Table 29. <i>Device-Wake.confirm</i> fields	
Description	Size
Status	1 byte

The Status shall be 0x00 if the ZigBee stack resumed successfully.

The application may then resume normal operation

Network-Sleep.request

The application may instruct other MailBox devices on the ZigBee network to sleep, even if they are supposedly always-on devices, by issuing a *Network-Sleep.request* containing the fields in Table 30.

Table 30. <i>Network-Sleep.request</i> fields	
Description	Size
True if <i>Broadcast</i>	1 bit
Broadcast Cluster (if <i>Broadcast</i>) Unicast Destination (otherwise)	2 bytes
Closedown period	2 bytes
Sleep period	3 bytes
Wakeup period	2 bytes

Network-Sleep.confirm

Upon receipt of the *Network-Sleep.request* primitive, the MailBox layer shall transmit a *Network Sleep* frame to all devices specified. It shall then confirm completion of the operation with a *Network-Sleep.confirm* primitive, as shown in Table 31.

Table 31. <i>Network-Sleep.confirm</i> fields	
Description	Size
Status	1 byte

The Status shall zero if the operation completed without error. In this case, the

Network-Sleep.indication

Upon receipt of a *Network Sleep* frame, the MailBox layer will verify the Application ID. If it is incorrect, the frame shall be ignored. If it is correct, it shall send a *Network-Sleep.indication* to its application layer, as shown in Table 32.

Table 32. <i>Network-Sleep.indication</i> fields	
Description	Size
Source address	2 bytes
Closedown period	2 bytes
Sleep period	3 bytes
Wakeup period	2 bytes

The application may elect to sleep provided the operation if any non-MailBox devices in the ZigBee network will not be affected. The sleep process is as follows:

The *Closedown period* is the duration, in seconds, from the moment *Network-Sleep.indication* was generated, that the application should continue to keep its MailBox layer active on and respond to messages if required. Failure of the parent to reply must be tolerated.

The MailBox layer may continue to transmit responses to transmissions received. The MailBox layer will continue to send *NMLE_SYNC.request* messages to its parent continues to do so during the *Closedown period* if required.

The application may not initiate any MailBox transmissions within the *Closedown* period.

The *Sleep period* is the duration, in seconds, from the moment the *Network Sleep* message was received, that the application may enter a sleep state. Prior to sleeping, a *Device-Sleep.request* should be issued to suspend and power down the ZigBee stack.

At the end of the *Sleep period*, the application shall wake up and issue a *Device-Wake.request* to power up and resume the ZigBee stack.

The *Wakeup period* is the duration, in seconds, immediately following the end of the *Sleep period*. During this period, the application shall enable the ZigBee stack and physical layer but not initiate any transmissions unless it first receives a transmission from another device.

Allowance must be made for the fact that a device may miss the *Network Sleep* broadcast. Should ZigBee operation time out, the application should attempt to rejoin the network using *Join.request*. Should rejoining fail on the first attempt, it should be retried frequently enough to re-sync in the event of a Network Sleep operation.

Present.request

The application may instruct other MailBox devices on the ZigBee network that is present by issuing a *Present.request* containing the fields in Table 33.

Table 33. <i>Present.request</i> fields	
Description	Size
True if <i>Broadcast</i>	1 bit
Broadcast Cluster (if <i>Broadcast</i>) Unicast Destination (otherwise)	2 bytes

Present.confirm

Upon receipt of the *Present.request* primitive, the MailBox layer shall transmit a *Present* frame to all

devices specified, without acknowledgement. It shall then confirm completion of the operation with a *Present.confirm* primitive, as shown in Table 34.

Table 34. <i>Present.confirm</i> fields	
Description	Size
Status	1 byte

The *Status* shall zero if the operation completed without error.

Present.indication

Upon receipt of a *Present* frame, the MailBox layer shall verify the Application ID and, if it matches, it shall send a *Present.indication* to its application layer, as shown in Table 35.

Table 35. <i>Present.indication</i> fields	
Description	Size
Device short address	2 bytes
Number of Functional Clusters	1 byte
Functional Cluster List	2 bytes each

The *Functional Cluster List* will not necessarily include the Mandatory Cluster 0x00. *Number of Functional Cluster* shall be the number of clusters in the list. Any null clusters in the list shall be ignored.

Redirect.request

The application may instruct other MailBox devices on the ZigBee network to change their Redirect Address by issuing a *Redirect.request* containing the fields in Table 36.

Table 36. <i>Redirect.request</i> fields	
Description	Size
Message recipient: <i>Broadcast</i>	1 bit
Message recipient: Broadcast Cluster (if <i>Broadcast</i>) Unicast Destination (otherwise)	2 bytes
New Redirect Address: <i>IsBroadcast</i>	1 bit
<i>IsAcknowledged</i>	1 bit
New Redirect Address: Broadcast Cluster (if <i>Broadcast</i>) Unicast Destination (otherwise)	2 bytes

Redirect.confirm

Upon receipt of the *Redirect.request* primitive, the MailBox layer shall transmit a *Present* frame to all devices specified, without acknowledgement. It shall then confirm completion of the operation with a *Redirect.confirm* primitive, as shown in Table 37.

Table 37. <i>Redirect.confirm</i> fields	
Description	Size
Status	1 byte

The *Status* shall zero if the operation completed without error.

Redirect.indication

Upon receipt of a *Redirect* frame, the MailBox layer shall verify the Application ID and, if it matches, it shall send a *Redirect.indication* to its application layer, with no associated fields. The MailBox layer will then update its Redirect Address to the value specified and use this as the destination address when subsequent transmissions are directed to the Redirect Address cluster, 0xFFFE.

Custom.request

The source application initiates the transfer of a packet of data by issuing a *Custom.request* containing the fields in Table 38.

Table 38. <i>Custom.request</i> fields	
Description	Size
True if <i>Broadcast</i>	1 bit
Broadcast Cluster (if <i>Broadcast</i>) Unicast Destination (otherwise)	2 bytes
Custom FID (0x80 – 0xFF)	1 byte
Application specific payload	≤64 bytes

Upon receipt of a *Custom.request*, the MailBox layer will transmit a Custom frame and then issue a *Custom.confirm*.

According to the application, the MailBox layer may generate its own *Custom* frames in addition to those requested by the application.

Custom.confirm

The Status of the transfer is conveyed back to the application with a *Data.confirm* containing the fields in Table 39.

Table 39. <i>Custom.confirm</i> fields	
Description	Size
Status	1 byte

Custom.indication

On receipt of a Custom frame, the destination MailBox layer shall verify the Application ID and ignore it if it is not correct. If it is correct, it will report the frame to its application layer by generating a *Custom.indication* primitive containing the fields in Table 40.

According to the application, the MailBox layer may alternatively process a Custom frame itself

rather than passing the generating a Custom.indication.

Table 40. Custom.indication fields	
Description	Size
Status 0x00 = Success	1 byte
True if Broadcast	1 bit
Source short address	2 bytes
Broadcast Cluster (if Broadcast)	2 bytes
Custom FID (0x80 – 0xFF)	1 byte
Data payload	≤64 bytes

MIB Attributes

The Mailbox maintains an Information Base of the following values:

The **Application ID** is the Application's 4-byte unique identifier.

The **AppEndpoint** is the ZigBee endpoint the MailBox shall use.

The **ClusterList** is the list of 2-byte Functional Clusters supported by the device.

The **MailBoxTimeout** period is the period after which it is assumed that an expected reply is not going to be received. The default value will be dependent on the stack profile.

PollRate Symbol periods between sleepy end device's polls to its parent for messages. Sleepy devices are required to do this while awake in order to pick up messages. Other devices (except the coordinator) may choose to do it simply to confirm that the parent is still there. Default value is 0x0080 for sleepy devices (approx half a second) and 0x0000 for other devices.

RetryDelay is a 2-byte variable which may be modified directly by the application. If it is zero, the application can accept data frames. If it is non-zero, it shall represent the minimum delay before a sending application may reattempt to transmit a packet, in $256 \times \text{RetryDelay}$ symbol periods.

RedirectAddr The destination to send messages to when the Redirect Cluster (0xFFFFE) is specified in a MDAR data request. If the first byte is zero, the second and third bytes shall be interpreted as a short address. If the first byte is nonzero, the second and third bytes shall be interpreted as a broadcast cluster. The value 0xFFFFF shall be interpreted as no address being specified and no transmission shall be made and the Data.confirm shall report success. Unlike other MIB values, this

value is volatile and is reset to 0xFFFFFFFF on startup.

SeqBufSiz is the number of records in the Sequence Buffer.

Development Support

FlexiPanel Ltd publishes free Sniffer firmware which parses MailBox frames which may prove useful. Example code snippets can also be provided to partners.

Please contact us if you have any comments or suggestions.

Revision list

Rev 1 – Initial release
Rev 2 – ZigBee Alliance Profile 0xC1EE assigned.

Contact details

The MailBox profile was developed by FlexiPanel Ltd:



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