WIREPAS

Things connected – Naturally

WP-RM-100 – Wirepas Mesh Dual-MCU API Reference Manual

Reference Manual

Version: v5.1A

Applies to Wirepas Mesh firmware release v5.0 onwards

This document describes the serial interface API provided by the Wirepas Mesh stack.

Confidential

Table of Contents

1. Introduction 4
1.1. Service Access Points 4
1.2. Primitive Types 4
1.3. Attributes 5
1.4. Serial Interface Specification 6
1.5. General Frame Format 6
1.6. Flow Control 7
1.7. UART Configuration 8
1.8. Endianness and Bit Order 8
1.9. Timing 8
1.10. CRC Calculation (CRC-16-CCITT) 8
2. Stack Service Specification 9
2.1. Node Addressing 10
2.2. Data Services (DSAP) 12
2.2.1. DSAP-DATA_TX Service 12
2.2.2. DSAP-DATA_RX Service 16
2.3. Management Services (MSAP) 17
2.3.1. INDICATION_POLL Service 18
2.3.2. MSAP-STACK_START Service 19
2.3.3. MSAP-STACK_STOP Service 20
2.3.4. MSAP-STACK_STATE Service 22
2.3.5. MSAP-APP_CONFIG_DATA_WRITE Service 22
2.3.6. MSAP-APP_CONFIG_DATA_READ Service 25
2.3.7. MSAP-APP_CONFIG_DATA_RX Service 26
2.3.8. MSAP-ATTRIBUTE_WRITE Service 27
2.3.9. MSAP-ATTRIBUTE_READ Service 28
2.3.10. MSAP-GET_NBORS Service 29
2.3.11. MSAP-SCAN_NBORS Service 31
2.3.12. MSAP-SINK_COST Service 32
2.3.13. MSAP-SCRATCHPAD Services 34
2.3.14. MSAP-NON-ROUTER LONG SLEEP (NRLS) Service 41
2.3.15. MSAP-MAX_MESSAGE_QUEUING Service 45
2.3.16. MSAP Attributes 47
2.4. Configuration Services (CSAP) 52
2.4.1. CSAP-ATTRIBUTE_WRITE Service 52
2.4.2. CSAP-ATTRIBUTE_READ Service 52
2.4.3. CSAP-FACTORY_RESET Service 53
2.4.4. CSAP Attributes 54
2.5. Response Primitives 63
2.6. Sequence Numbers 63

3. Common Use Cases....64

3.1. Required Configuration 64

4. Annex A: Additional CRC Information....65

4.1. Example CRC Implementation....65

4.2. CRC Test Vectors 66

5. References....67

1. Introduction

The Wirepas Mesh stack (hereafter referred to as the “stack”) provides services for the application layer (hereafter referred to as the “application”). The services are exposed via Service Access Points (“SAPs”). The SAPs are divided into Data SAP (DSAP), Management SAP (MSAP), and Configuration SAP (CSAP). The SAP services are provided in the form of primitives and SAP data is exposed as attributes.

All field lengths are in octets (i.e. units of eight bits), unless otherwise stated.

1.1. Service Access Points

The SAPs provide the following general services:

- **DSAP**: Provides methods for data transfer to and from the stack (and the network)
- **MSAP**: Provides methods for transferring stack management information and reading/writing management attributes. Management attributes provide information of the run-time state of the stack and are valid only when the stack is running.
- **CSAP:** Provides methods for reading/writing stack configuration attributes. Configuration attributes can only be written when the stack is stopped.

Currently, the SAPs are realized as a Universal Asynchronous Receiver/Transmitter (UART) serial interface.

1.2. Primitive Types

The primitives are divided into four classes: request, confirm, indication, and response. The general usage of the primitives is as follows (Also see Figure 1):

- A **request** is issued by the application when it wants to use a stack service.
- A **confirm** is a reply from the stack to the request issued by the application.
- An **indication** is issued by the stack when it has data/information it wants to send to the application. In the point of view of the application, indications are asynchronous messages from the stack.
- A **response** is a reply from the application to the indication issued by the stack.

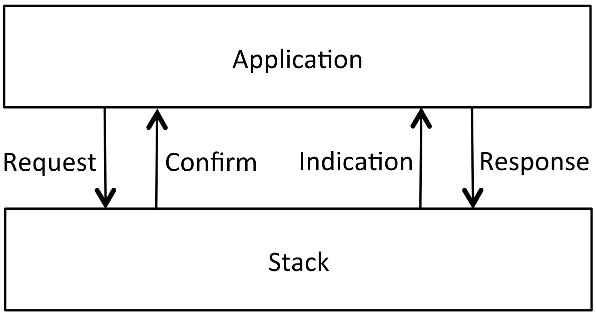


Figure 1. Primitive usage in the communication between the application and the stack

Three different use cases can be identified for the above primitives:

1. Application issues commands to the stack or needs to send data/information.
    a. Application issues the appropriate **request** primitive.
    b. The stack responds with corresponding **confirm** primitive to acknowledge the request.

2. Application queries data/information from the stack and the stack responds immediately:
    a. Application issues the appropriate **request** primitive.
    b. The stack responds with corresponding **confirm** primitive containing the requested data/information.

3. The stack needs to send asynchronous data/information to the application:
    a. The stack generates appropriate **indication(s)**.
    b. The stack asserts the Interrupt ReQuest ("IRQ") signal to notify the application that it has one or more pending indications.
    c. The application queries the indications from the stack and acknowledges every indication with corresponding **response** primitive.

Note 1: Some application requests may generate an immediate response with which the stack informs that the request has been taken for processing and in addition optional indication with which the stack informs that the request has actually been processed.

Note 2: The stack indications are always notified via IRQ and can be queried by the application. The stack never sends data/information to the application on its own without the application explicitly requesting it. This enables the application to have full control over the communication between the application and the stack, and offers full flexibility on the application architecture (interrupt-based/polling) and scheduling (application can sleep and run its own tasks when it wants to and communicate with the stack when it wants to). In an extreme case, e.g. when application MCU pin count is too low, the IRQ signal can even be omitted and the indication queries can be sent periodically, though this implementation is not the most energy-efficient nor provides lowest delay depending on the query interval.

1.3. Attributes

Attributes are small pieces of data that affect the way the stack works, or are used to inform the application of the state of the stack. Before the stack can be started in normal operation, a few critical attributes need to be configured properly (see section 3.1).

Attributes can either be read-only, readable and writable, or write-only. The attributes can also be persistent or non-persistent. If the attribute is persistent, its value will be retained over device power downs and stack stops, i.e. the value of an attribute is stored in non-volatile memory. Otherwise, the attribute value will be lost when the device is powered down or the stack stopped.

Note: Although there are no strict restrictions on how often a persistent variable can be updated by the application layer, each update causes a tiny bit of wear on the non-volatile memory. If a persistent variable is to be updated periodically, updating it less often than once every 30 minutes is recommended.

1.4. Serial Interface Specification

The physical interface between the application MCU and stack MCU is Universal Asynchronous Receiver/Transmitter (“UART”), colloquially called a serial port. The data is exchanged in frames.

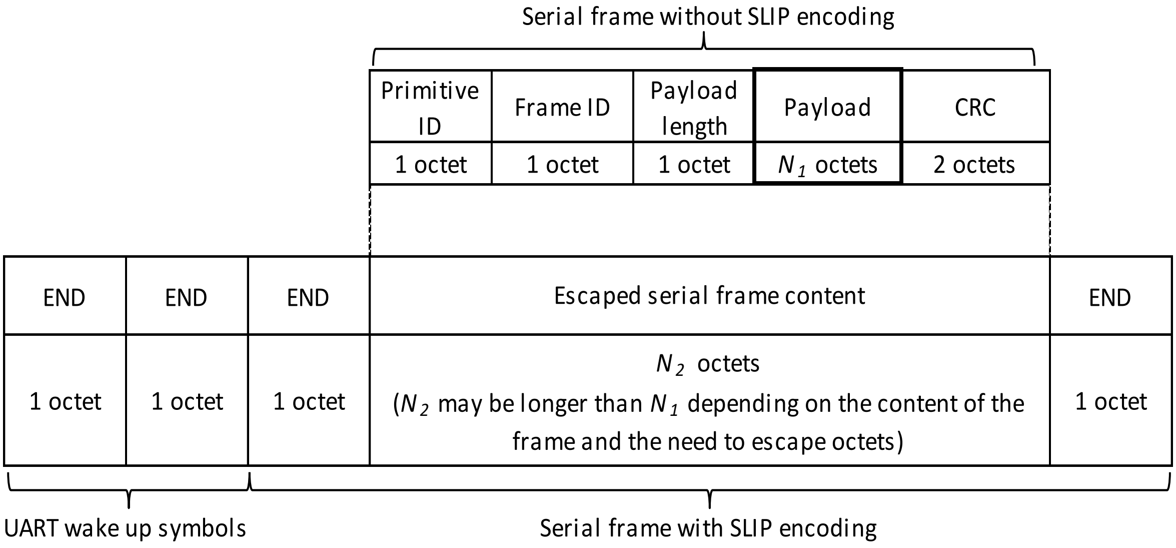
1.5. General Frame Format

The serial frames have similar frame separation mechanism as in SLIP (RFC 1055). SLIP framing works as follows:

- Two octet values are reserved: 0xC0, called “END” and 0xDB, called “ESC”.
- A frame begins and ends with octet 0xC0 (END).
- Any octet of value 0xC0 (END) within the frame is encoded as 0xDB (ESC), 0xDC.
- Any octet of value 0xDB (ESC) within the frame is encoded as 0xDB (ESC), 0xDD.
- Any other octet is passed through as-is.

In addition, two additional END octets are used to wake up the stack side UART when starting communication. These END octets are present only when communicating towards the stack UART. The stack UART will transmit a single END octet in the beginning of a frame.

The general format of the serial frame is presented in Figure 2. Note that the different primitives and corresponding content of the payload (thick border in Figure 2) are specified in section 2. The meaning of the different frame fields is described in Table 1.


Serial frame without SLIP encoding
UART wake up symbols
Serial frame with SLIP encoding

Primitive ID	Frame ID	Payload length	Payload	CRC
1 octet	1 octet	1 octet	N_1 octets	2 octets

END	END	END	Escaped serial frame content	END
1 octet	1 octet	1 octet	N_2 octets (N_2 may be longer than N_1 depending on the content of the frame and the need to escape octets)	1 octet

Figure 2. General format of the serial frame

Table 1. General serial frame fields

Field	Size	Description
END	1	Frame separator, octet 0xC0. Starts and ends a SLIP encoded frame. In addition two extra END-octets are used to wake up the stack side UART when starting communication.
Primitive ID	1	The identified of the used primitive. Different primitives and their primitive identifiers are specified in section 2. As a general rule: • Initiating primitives (request-primitives from the application side and indication-primitives from the stack side) have always the most significant bit set to 0. • Responding primitives (confirm-primitives from the stack side and response-primitives from the application side) always have the most significant bit set to 1. • Confirm.primitive_id = 0x80 | request.primitive_id • Response.primitive_id = 0x80 | indication.primitive_id
Frame ID	1	Frame identifier. The initiating peer decides the ID and responding peer uses the same value in the response frame: • The application decides the Frame ID for a request-primitive and the stack sends corresponding confirm-primitive with the same Frame ID. • The stack decides the Frame ID for an indication-primitive and the application sends corresponding response-primitive with the same Frame ID.
Payload length	1	The following payload length in octets, excluding the CRC octets.
Payload	N_1	The payload of the frame, depends on the primitive in question. Different primitives and corresponding content of the payload are specified in section 2.
CRC	2	Checksum over the whole frame that has not been SLIP encoded, excluding the CRC octets. When receiving a frame, the SLIP encoding is removed and the CRC is calculated over the decoded frame. When sending a frame, the CRC is calculated first and SLIP encoding is employed after that.

Note: These fields are used only locally for the communication between the application MCU and the stack. They are not actually transmitted on the network.

1.6. Flow Control

The application MCU is the master in the communication between the application and the stack.

The stack UART receiver is enabled by two wake up symbols (as described in section 1.5) and any octets received via UART are processed in an Interrupt Service Routine (ISR). Thus, no serial interface flow control is required when communicating to the stack (flow control may be needed in upper level if the stack memory runs low due to congestion).

Communication is always initiated by the application MCU. Thus, no serial interface flow control is required in the application direction either.

The stack informs pending indications via an IRQ signal. The IRQ signal is active low. When the stack has pending indications, the IRQ is asserted, i.e. the IRQ signal is pulled down. When the stack does not have pending indications, the IRQ is not asserted, i.e. the IRQ signal is held high.

The usage of request-confirm and indication-response pairs should always be atomic. This means, that a new request should not be sent before a confirmation is received for a previous request (application initiated communication) and a new indication should not be sent before a response is received for a previous indication (stack initiated communication).

1.7. UART Configuration

When the stack MCU starts up, it chooses a UART configuration by reading the state of dedicated (hardware dependent) configuration pins. Data bit, stop bit and parity configuration is fixed, but the UART bitrate has two options, as presented in Table 2.

Table 2. UART configuration

Parameter	Value
Baud rate	115200 bps or 125000 bps
Number of data bits	8
Parity	No parity bit
Number of stop bits	1

1.8. Endianness and Bit Order

Multi-octet fields are transferred least significant octet first (i.e. little-endian).

Octets are transferred most significant bit first.

1.9. Timing

There is a reception timeout for received UART frames. A transmission of complete API frame to the stack MCU shall take no longer than 100 ms.

1.10. CRC Calculation (CRC-16-CCITT)

The used CRC type is CRC-16-CCITT. See Annex A for example implementation and test vectors.

2. Stack Service Specification

The different services provided by the stack are specified in this section. The specification includes description of usage, primitives, and frame formats of the services. Table 3 list all the primitives and their primitive IDs.

Table 3. All primitives and their primitive IDs

SAP	Primitive	Primitive ID
DSAP	DSAP-DATA_TX.request	0x01
	DSAP-DATA_TX.confirm	0x81
	DSAP-DATA_TX_TT.request	0x1F
	DSAP-DATA_TX_TT.confirm	0x9F
	DSAP-DATA_TX.indication	0x02
	DSAP-DATA_TX.response	0x82
	DSAP-DATA_RX.indication	0x03
	DSAP-DATA_RX.response	0x83
MSAP	MSAP-INDICATION_POLL.request	0x04
	MSAP-INDICATION_POLL.confirm	0x84
	MSAP-STACK_START.request	0x05
	MSAP-STACK_START.confirm	0x85
	MSAP-STACK_STOP.request	0x06
	MSAP-STACK_STOP.confirm	0x86
	MSAP-STACK_STATE.indication	0x07
	MSAP-STACK_STATE.response	0x87
	MSAP-APP_CONFIG_DATA_WRITE.request	0x3A
	MSAP-APP_CONFIG_DATA_WRITE.confirm	0xBA
	MSAP-APP_CONFIG_DATA_READ.request	0x3B
	MSAP-APP_CONFIG_DATA_READ.confirm	0xBB
	MSAP-APP_CONFIG_DATA_RX.indication	0x3F
	MSAP-APP_CONFIG_DATA_RX.response	0xBF
	MSAP-NRLS.request	0x40
	MSAP-NRLS.confirm	0xC0
	MSAP-NRLS_STOP.request	0x41
	MSAP-NRLS_STOP.confirm	0xC1
	MSAP-NRLS_STATE_GET.request	0x42
	MSAP-NRLS_STATE_GET.response	0xC2
	MSAP-NRLS_GOTOSLEEP_INFO.request	0x4C
	MSAP-NRLS_GOTOSLEEP_INFO.response	0xCC
	MSAP-ATTRIBUTE_WRITE.request	0x0B
	MSAP-ATTRIBUTE_WRITE.confirm	0x8B
	MSAP-ATTRIBUTE_READ.request	0x0C
	MSAP-ATTRIBUTE_READ.confirm	0x8C
	MSAP-GET_NBORS.request	0x20
	MSAP-GET_NBORS.confirm	0xA0
	MSAP-SCAN_NBORS.request	0x21
	MSAP-SCAN_NBORS.confirm	0xA1
	MSAP-SCAN_NBORS.indication	0x22
	MSAP-SCAN_NBORS.response	0xA2
	MSAP-SINK_COST_WRITE.request	0x38
	MSAP-SINK_COST_WRITE.confirm	0xB8
	MSAP-SINK_COST_READ.request	0x39
	MSAP-SINK_COST_READ.confirm	0xB9
	MSAP-SCRATCHPAD_START.request	0x17
	MSAP-SCRATCHPAD_START.confirm	0x97
	MSAP-SCRATCHPAD_BLOCK.request	0x18

SAP	Primitive	Primitive ID
	MSAP-SCRATCHPAD_BLOCK.confirm	0x98
	MSAP-SCRATCHPAD_STATUS.request	0x19
	MSAP-SCRATCHPAD_STATUS.confirm	0x99
	MSAP-SCRATCHPAD_UPDATE.request	0x1A
	MSAP-SCRATCHPAD_UPDATE.confirm	0x9A
	MSAP-SCRATCHPAD_CLEAR.request	0x1B
	MSAP-SCRATCHPAD_CLEAR.confirm	0x9B
	MSAP-MAX_QUEUE_TIME_WRITE.request	0x4F
	MSAP-MAX_QUEUE_TIME_WRITE.confirm	0xCF
	MSAP-MAX_QUEUE_TIME_READ.request	0x50
	MSAP-MAX_QUEUE_TIME_READ.confirm	0xD0
CSAP	CSAP-ATTRIBUTE_WRITE.request	0x0D
	CSAP-ATTRIBUTE_WRITE.confirm	0x8D
	CSAP-ATTRIBUTE_READ.request	0x0E
	CSAP-ATTRIBUTE_READ.confirm	0x8E
	CSAP-FACTORY_RESET.request	0x16
	CSAP-FACTORY_RESET.confirm	0x96

Note: The general framing follows the format described in section 1.5. For clarity, the figures of the frames presented in this section also include the general frame fields (Primitive ID, Frame ID, Payload length, and CRC), but their descriptions are omitted as they are already explained in section 1.5.

2.1. Node Addressing

The Wirepas Mesh Dual-MCU API services use a 32-bit address to indicate sources and destinations of packets.

Two special addresses have been reserved. First, address 0x0000 0000 (zero) or 0xFFFFFFFE (4 294 967 294) is used as the *anySink* address which identifies that the source or the destination of a packet is an unspecified sink on the network. The highest address 0xFFFF FFFF is used as the *broadcast* address. It is used to transmit a downlink packet to all nodes on the network from a sink.

Nodes are not allowed to use these two special addresses as their own address. Similarly, addresses in multicast address space cannot be used as own address.

The addresses are summarized in Table 4.

Table 4. Addressing summary

Address type	Valid address space	Description
Unicast	0x00000001- 0x7FFFFFFF (1 – 2 147 483 647) and 0x81000000- 0xFFFFFFFD (2 164 260 864 – 4 294 967 293)	Valid unicast addresses. Each node on the network must have one of these addresses set as its address. Two or more devices with identical addresses should never be present on a network.
Broadcast	0xFFFF FFFF (4 294 967 295)	Broadcast address with which a packet is delivered to all nodes on the network
AnySink	0xFFFFFFFE (4 294 967 294) or 0x00000000 (0)	Address which identifies that the source or the destination of a packet is an unspecified sink on the network With current tree routing, only nodes may use this as the destination address when sending packets. These addresses are reserved as Wirepas reserved addresses and cannot be used as addresses for any nodes in the network.
Multicast	0x80000000- 0x80FFFFFF (2 147 483 648 – 2 164 260 863)	Packet is delivered to the group of nodes. Group may contain 0 or more nodes. Each node may belong to 0 or more groups. The lowest 24 bits contain the actual group address and highest bit is an indication that message is sent to that group.

2.2. Data Services (DSAP)

The data services are used to transmit/receive application data via the network.

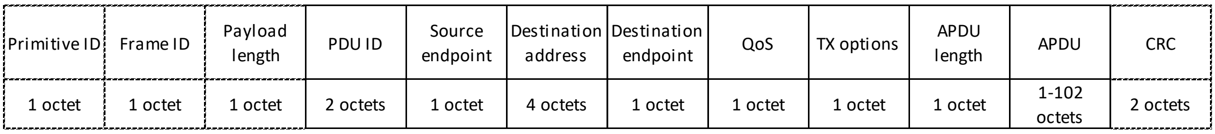
2.2.1. DSAP-DATA_TX Service

The DSAP-DATA_TX service is used to transport APDUs from the application to the stack. The stack transmits the APDUs to other node(s) on the network, according to set parameters. The DSAP-DATA_TX service includes the following primitives:

- DSAP-DATA_TX.request
- DSAP-DATA_TX.confirm
- DSAP-DATA_TX_TT.request
- DSAP-DATA_TX_TT.confirm
- DSAP-DATA_TX.indication
- DSAP-DATA_TX.response (All response primitives have the same format, see section 2.5)

2.2.1.1. DSAP-DATA_TX.request

The DSAP-DATA_TX.request is issued by the application when it wants to send data. The DSAP-DATA.request frame is depicted in Figure 3.



Primitive ID	Frame ID	Payload length	PDU ID	Source endpoint	Destination address	Destination endpoint	QoS	TX options	APDU length	APDU	CRC
1 octet	1 octet	1 octet	2 octets	1 octet	4 octets	1 octet	1 octet	1 octet	1 octet	1-102 octets	2 octets

Figure 3. DSAP-DATA_TX.request frame

The DSAP-DATA_TX.request frame fields (solid border in the figure) are explained in Table 5.

Table 5. DSAP-DATA_TX.request frame fields

Field Name	**Size**	**Valid Values**	**Description**
PDUID	2	0 – 65534	PDU identifier decided by the application The PDU ID can be used to keep track of APDUs processed by the stack as the same PDU ID is contained in a corresponding DSAP-DATA_TX.indication sent by the stack to the application. E.g. the application can keep the PDU in its own buffers until the successful transmission is indicated by the stack in DSAP-DATA_TX.indication. PDU ID 65535 (0xFFFF) is reserved and should not be used. Also see Note 1.
SourceEndpoint	1	0 – 239	Source endpoint number Also see Note 2.
DestinationAddress	4	0 – 4294967295	Destination node address Also see Note 3.
DestinationEndpoint	1	0 – 239	Destination endpoint number Also see Note 2.

Field Name	Size	Valid Values	Description
QoS	1	0 or 1	Quality of service class to be used. The different values are defined as follows: 0 = Use traffic class 0, i.e. normal priority 1 = Use traffic class 1, i.e. high priority.
TXOptions	1	00xx xxxx (bitfield, where x can be 0 or 1)	The TX options are indicated as a bit field with individual bits defined as follows: Bit 0 = 0: Do not generate DSAP-DATA_TX.indication Bit 0 = 1: Generate DSAP-DATA_TX.indication Bit 0 is used to register for receiving a DSAP-DATA_TX.indication after the PDU has been successfully transmitted to next hop node or cleared from the PDU buffers due to timeout or congestion. Also see Note 1. Bit 1 = 1, Use unacknowledged CSMA-CA transmission method Bit 1 = 0, Use normal transmission method. See Note 4. Bits 2-5: Hop limit. Maximum number of hops executed for packet to reach the destination. See Note 5. Bits 6-7: Reserved Here, bit 0 is the least significant bit and bit 7 is the most significant bit.
APDULength	1	1 – 102	The length of the following APDU in octets
APDU	1 – 102	-	Application payload

Note 1: These fields are used only locally for the communication between the application layer and the stack. They are not actually transmitted on the network. Only 16 requests where generation of DSAP-DATA_TX.indication is active are allowed at the same time. Without generation of the indication, there is room for plenty of more requests simultaneously.

Note 2: The endpoint numbers are used to distinguish different application channels. E.g. if the device has multiple sensors it could use different endpoint numbers for APDUs containing data from different sensors or different endpoint numbers for applications with different functionality.

Endpoints 240 – 255 are reserved for Wirepas Mesh stack internal use.

Note 3: Note that a broadcast will only be transmitted (downlink) to the nodes directly under the sink's routing tree. To reach all nodes on the network, it is necessary to send the broadcast from all sinks. All devices can send traffic to themselves (loopback) by using their own address as destination.

Note 4: The unacknowledged CSMA-CA transmission method can be used in a mixed network (i.e. network consisting of both CSMA-CA and TDMA devices) by CSMA-CA device originated packets transmission only to CSMA-CA devices. The purpose of this method is to avoid a performance bottleneck by NOT transmitting to TDMA devices. Also, if used with sink-originated transmissions (by CSMA-CA mode sinks), the throughput is better when compared to a 'normal' transmission, however there is some penalty in reliability (due to unacknowledged nature of transmission).

Note 5: Hop limit sets the upper value to the number of hops executed for packet to reach the destination. By using hop limiting, it is possible to limit the distance how far the packet is transmitted to and avoiding causing unnecessary traffic to network. Hop count value of 0 is used to disable the hop limiting. Hop limiting value does **not** have any impact when using *AnySink* address as destination node address but is discarded.

2.2.1.2. DSAP-DATA_TX.confirm

The DSAP-DATA_TX.confirm is issued by the stack as a response to the DSAP-DATA_TX.request. The DSAP-DATA_TX.confirm frame is depicted in Figure 4.

Primitive ID	Frame ID	Payload length	PDU ID	Result	Capacity	CRC
1 octet	1 octet	1 octet	2 octets	1 octet	1 octet	2 octets

Figure 4. DSAP-DATA_TX.confirm frame

The DSAP-DATA_TX.confirm frame fields (solid border in the figure) are explained in

Table 6.

Table 6. DSAP-DATA_TX.confirm frame fields

Field Name	Size	Valid Values	Description
PDUID	2	0 – 65534	PDU identifier set by the application in the corresponding DSAP-DATA_TX.request This field is only used for data TX requests where an indication is requested, i.e. TX options bit 0 is set (see field TXOptions in section 2.2.1.1). If no indication is requested, the value of this field is undefined.
Result	1	0 – 10	The return result of the corresponding DSAP-DATA_TX.request. The different values are defined as follows: 0 = Success: PDU accepted for transmission 1 = Failure: Stack is stopped 2 = Failure: Invalid QoS-parameter 3 = Failure: Invalid TX options-parameter 4 = Failure: Out of memory 5 = Failure: Unknown destination address 6 = Failure: Invalid APDU length-parameter 7 = Failure: Cannot send indication 8 = Failure: PDUID is already in use 9 = Failure: Invalid src/dest end-point 10 = Failure: Access denied (see section 2.4.4.21)
Capacity	1	-	Number of PDUs that still can fit in the PDU buffer (see section 2.3.16.3 for details)

2.2.1.3. DSAP-DATA_TX_TT.request

The DSAP-DATA_TX_TT.request is identical to the DSAP-DATA_TX.request, except there is one extra field for setting buffering delay to an initial no-zero value. The DSAP-DATA.request frame is depicted in Figure 5.

Primitive ID	Frame ID	Payload length	PDU ID	Source endpoint	Destination address	Destination endpoint	QoS	TX options	Buffering delay	APDU length	APDU	CRC
1 octet	1 octet	1 octet	2 octets	1 octet	4 octets	1 octet	1 octet	1 octet	4 octets	1 octet	1-102 octets	2 octets

Figure 5. DSAP-DATA_TX_TT.request frame

The DSAP-DATA_TX_TT.request frame fields (solid border in the figure) are explained in Table 7.

Table 7. DSAP-DATA_TX_TT.request frame fields

Field Name	Size	Valid Values	Description
PDUID	2	0 – 65534	See description in chapter 2.2.1.1.
SourceEndpoint	1	0 – 239	See description in chapter 2.2.1.1.
DestinationAddress	4	0 – 4294967295	See description in chapter 2.2.1.1.
DestinationEndpoint	1	0 – 239	See description in chapter 2.2.1.1.
QoS	1	0 or 1	See description in chapter 2.2.1.1.
TXOptions	1	00xx xxxx (bitfield, where x can be 0 or 1)	See description in chapter 2.2.1.1.
BufferingDelay	4	0 – 4 294 967 295	The time the PDU has been in the application buffers before it was transmitted over API. Expressed in units of 1/128th of a second.
APDULength	1	1 – 102	See description in chapter 2.2.1.1.

Field Name	Size	Valid Values	Description
APDU	1 – 102	-	See description in chapter 2.2.1.1.

2.2.1.4. DSAP-DATA_TX_TT.confirm

The DSAP-DATA_TX_TT.confirm is issued by the stack as a response to the DSAP-DATA_TX_TT.request. It is identical to DSAP-DATA_TX.confirm, explained in section 2.2.1.2.

2.2.1.5. DSAP-DATA_TX.indication

The DSAP-DATA_TX.indication is issued by the stack as an asynchronous reply for the DSAP-DATA_TX.request after the APDU of the corresponding DSAP-DATA_TX.request is successfully transmitted to next hop node or cleared from stack buffers due to timeout or congestion. The DSAP-DATA_TX.indication is sent only if the application registers it in the corresponding DSAP-DATA_TX.request's TX options parameter. The DSAP-DATA_TX.indication frame is depicted in Figure 6.

Primitive ID	Frame ID	Payload length	Indication status	PDU ID	Source endpoint	Destination address	Destination endpoint	Buffering delay	Result	CRC
1 octet	1 octet	1 octet	1 octet	2 octets	1 octet	4 octets	1 octet	4 octets	1 octet	2 octets

Figure 6. DSAP-DATA_TX.indication frame

The DSAP-DATA_TX.indication frame fields (solid border in the figure) are explained in Table 8.

Table 8. DSAP-DATA_TX.indication frame fields

Field Name	Size	Valid Values	Description
IndicationStatus	1	0 or 1	0 = No other indications queued 1 = More indications queued
PDUID	2	0 – 65534	PDU identifier set by the application in the corresponding DSAP-DATA_TX.request
SourceEndpoint	1	0 – 239	Source endpoint number
DestinationAddress	4	0 – 4294967295	Destination node address set by the application in the corresponding DSAP-DATA_TX.request
DestinationEndpoint	1	0 – 239	Destination endpoint number
BufferingDelay	4	-	The time the PDU has been in the stack buffers before it was transmitted. Reported in units of *BufferingDelay / 128* seconds i.e. *BufferingDelay * 7.8125* milliseconds.
Result	1	0 or 1	The return result of the corresponding DSAP-DATA_TX.request. The different values are defined as follows: 0 = Success: PDU was successfully sent 1 = Failure: PDU was discarded

2.2.2. DSAP-DATA_RX Service

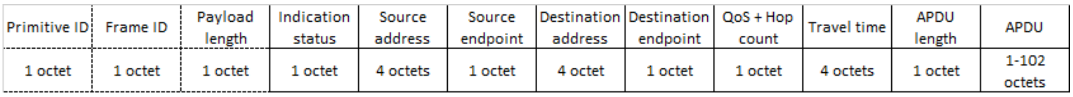
The DSAP-DATA_RX service supports the transport of received APDUs from the stack to the application layer. The DSAP-DATA_RX service includes the following primitives:

- DSAP-DATA_RX.indication

- DSAP-DATA_RX.response (All response primitives have the same format, see section 2.5)

2.2.2.1. DSAP-DATA_RX.indication

The DSAP-DATA_RX.indication is issued by the stack when it receives data from the network destined to this node. The DSAP-DATA_RX.indication frame is depicted in Figure 7.



Primitive ID	Frame ID	Payload length	Indication status	Source address	Source endpoint	Destination address	Destination endpoint	QoS + Hop count	Travel time	APDU length	APDU
1 octet	1 octet	1 octet	1 octet	4 octets	1 octet	4 octet	1 octet	1 octet	4 octets	1 octet	1-102 octets

Figure 7. DSAP-DATA_RX.indication frame

The DSAP-DATA_RX.indication frame fields (thick border in the figure) are explained in Table 9.

Table 9. DSAP-DATA_RX.indication frame fields

Field Name	Size	Valid Values	Description
IndicationStatus	1	0 or 1	0 = No other indications queued 1 = More indications queued
SourceAddress	4	0 – 4294967295	Source node address
SourceEndpoint	1	0 – 239	Source endpoint number
DestinationAddress	4	0 – 4294967295	Destination node address
DestinationEndpoint	1	0 – 239	Destination endpoint number
QoS + Hop count	1	0 – 255	Bits 0-1 (LSB): Quality of service class to be used. The different values are defined as follows: 0 = Use traffic class 0, i.e. normal priority 1 = Use traffic class 1, i.e. high priority Bits 2-7: Hop count: how many hops were used to transmit the data to the destination (1-n hops) For example, value 0x29 (0b00101001) tells that high priority data was received and ten hops were used to transmit data to the destination.
TravelTime	4	-	Travel time of the PDU on the network. Reported in units of *TravelTime / 128* seconds i.e. *TravelTime * 7.8125* milliseconds.
APDULength	1	-	The length of the following APDU in octets
APDU	1 – 102	-	Application payload

2.3. Management Services (MSAP)

The management services are used to control the stack at run-time, as well as read and write run-time parameters of the stack.

2.3.1. INDICATION_POLL Service

For enabling variety of application architectures, the application may poll the stack indications when it is most convenient. This is enabled by the indication IRQ with MSAP-INDICATION_POLL service. The MSAP-INDICATION_POLL service includes the following primitives:

- MSAP-INDICATION_POLL.request
- MSAP-INDICATION_POLL.confirm

The MSAP-INDICATION_POLL service is used to query indications from the stack. This mechanism is used for all indications independent of the service.

The basic flow for receiving indications from the stack goes as follows:

1. The stack asserts the IRQ signal to indicate that it has pending indication(s) that it wants to deliver to the application. For hardware-specific information on the IRQ signal, see the appropriate hardware reference manual.
2. The application sends MSAP-INDICATION_POLL.request to query for the indication(s).
3. The stack responds with MSAP-INDICATION_POLL.confirm to indicate that it will start sending pending indications.
4. The stack sends a pending indication. The individual indication format depends on the service that has issued the indication and follows the indication formats specified in this document.
5. The application sends a response to acknowledge the indication. In the response, the application also indicates if it wants to receive another pending indication.
6. If the response frame indicated that the application wants to receive another indication and there are still pending indications: Go to step 4.

The indication exchange stops if a) there are no more pending indications (in which case the stack de-asserts the IRQ), or b) the application indicates in a response that it does not want to receive more indications at the moment (in which case pending indications, if there are any, can be queried later).

Note: If there are no pending indications when the application issues a MSAP-INDICATION_POLL.request (i.e. the request is issued, but IRQ signal is not asserted), the stack replies only with MSAP-INDICATION_POLL.confirm and informs that there are no pending indications at the moment.

2.3.1.1. MSAP-INDICATION_POLL.request

The MSAP-INDICATION_POLL.request is issued by the application layer when it wants to query stack indications. The MSAP-INDICATION_POLL.request frame is depicted in Figure 8.

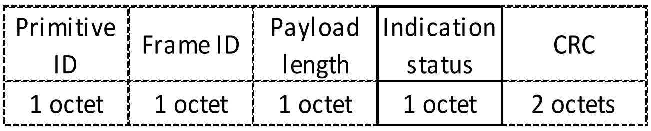
Primitive ID	Frame ID	Payload length	CRC
1 octet	1 octet	1 octet	2 octets

Figure 8. MSAP-INDICATION_POLL.request frame

The MSAP-INDICATION_POLL.request frame does not contain any payload.

2.3.1.2. MSAP-INDICATION_POLL.confirm

The MSAP-INDICATION_POLL.confirm is issued by the stack as a response to the MSAP-INDICATION_POLL.request. The MSAP-INDICATION_POLL.confirm frame is depicted in Figure 9.



Primitive ID	Frame ID	Payload length	Indication status	CRC
1 octet	1 octet	1 octet	1 octet	2 octets

Figure 9. MSAP-INDICATION_POLL.confirm frame

The MSAP-INDICATION_POLL.confirm frame fields (solid border in the figure) are explained in Table 10.

Table 10. MSAP-INDICATION_POLL.confirm frame fields

Field Name	Size	Valid Values	Description
Result	1	0 or 1	The return result of the corresponding MSAP-INDICATION_POLL.request. The different values are defined as follows: 1 = Pending indications exist and stack will start sending the indication(s) 0 = No pending indications

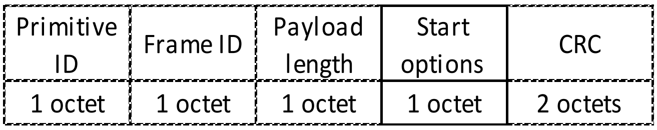
2.3.2. MSAP-STACK_START Service

The stack can be started using the MSAP-STACK_START service. The MSAP-STACK_START service includes the following primitives:

- MSAP-STACK_START.request
- MSAP-STACK_START.confirm

2.3.2.1. MSAP-STACK_START.request

The MSAP-STACK_START.request issued by the application layer when the stack needs to be started. The MSAP-STACK_START.request frame is depicted in Figure 10.



Primitive ID	Frame ID	Payload length	Start options	CRC
1 octet	1 octet	1 octet	1 octet	2 octets

Figure 10. MSAP-STACK_START.request frame

The MSAP-STACK_START.request frame fields (solid border in the figure) are explained in

Table 11.

Table 11. MSAP-STACK_START.request frame fields

Field Name	Size	Valid Values	Description
StartOptions	1	0000 000x (where x can be 0 or 1)	The stack start options are indicated as a bit field with individual bits defined as follows: Bit 0 = 0: Start stack with auto-start disabled, see Note, below Bit 0 = 1: Start stack with auto-start enabled Bit 1: Reserved Bit 2: Reserved Bit 3: Reserved Bit 4: Reserved Bit 5: Reserved Bit 6: Reserved Bit 7: Reserved , where bit 0 is the least significant bit and bit 7 is the most significant bit.

Note: For more information on the auto-start feature, see section 2.3.12 (MSAP *mAutostart*-attribute). The setting in bit 0 is automatically written to the MSAP auto-start attribute.

2.3.2.2. MSAP-STACK_START.confirm

The MSAP-STACK_START.confirm issued by the stack as a response to the MSAP-STACK_START.request. The MSAP-STACK_START.confirm frame is depicted in Figure 11.

Primitive ID	Frame ID	Payload length	Result	CRC
1 octet	1 octet	1 octet	1 octet	2 octets

Figure 11. MSAP-STACK_START.confirm frame

The MSAP-STACK_START.confirm frame fields are explained in Table 12.

Table 12. MSAP-STACK_START.confirm frame fields

Field Name	Size	Valid Values	Description
Result	1	000x xxxx (where x can be 0 or 1)	The return result of the corresponding MSAP-STACK_START.request. The result is indicated as a bit field with individual bits defined as follows: 0x00: Success: Stack started Bit 0 = 1: Failure: Stack remains stopped Bit 1 = 1: Failure: Network address missing Bit 2 = 1: Failure: Node address missing Bit 3 = 1: Failure: Network channel missing Bit 4 = 1: Failure: Role missing Bit 5 = 1: Failure: Application configuration data missing (valid only on sink device) Bit 6: Reserved Bit 7 = 1: Failure: Access denied (see section 2.4.4.21) , where bit 0 is the least significant bit and bit 7 is the most significant bit.

2.3.3. MSAP-STACK_STOP Service

The stack can be stopped using the MSAP-STACK_STOP service. The MSAP-STACK_STOP service includes the following primitives:

- MSAP-STACK_STOP.request
- MSAP-STACK_STOP.confirm

2.3.3.1. MSAP-STACK_STOP.request

The MSAP-STACK_STOP.request is issued by the application layer when the stack needs to be stopped. Stopping the stack will cause the firmware to reboot. This has the following side effects:

- All buffered data will be lost, including data being routed from other nodes
- If there is a new OTAP scratchpad that has been marked to be processed, the bootloader will process it, e.g. update the stack firmware (see section 2.3.13)

Note: A successful MSAP-STACK_STOP.request sets the MSAP auto-start attribute to disabled. For more information on the auto-start feature, see section 2.3.12 (MSAP *mAutostart*-attribute).

The MSAP-STACK_STOP.request frame is depicted in Figure 12.

Primitive ID	Frame ID	Payload length	CRC
1 octet	1 octet	1 octet	2 octets

Figure 12. MSAP-STACK_STOP.request frame

The MSAP-STACK_STOP.request frame does not contain any payload.

2.3.3.2. MSAP-STACK_STOP.confirm

The MSAP-STACK_STOP.confirm issued by the stack as a response to the MSAP-STACK_STOP.request. The MSAP-STACK_STOP.confirm frame is depicted in Figure 13.

Primitive ID	Frame ID	Payload length	Result	CRC
1 octet	1 octet	1 octet	1 octet	2 octets

Figure 13. MSAP-STACK_STOP.confirm frame

The MSAP-STACK_STOP.confirm frame fields are explained in Table 13.

Table 13. MSAP-STACK_STOP.confirm frame fields

Field Name	Size	Valid Values	Description
Result	1	0, 1 or 128	The return result of the corresponding MSAP-STACK_STOP.request. The different values are defined as follows: 0 = Success: Stack stopped 1 = Failure: Stack already stopped 128 = Failure: Access denied (see section 2.4.4.21)

Note: After device has sent the MSAP-STACK_STOP.confirm message, device is rebooted. This takes a while. During the rebooting, the device does not respond to messages. One example of how to detect when the rebooting has been done is to issue read-only commands to the device and when it responses to such, the rebooting has been performed. For example, use MSAP-ATTRIBUTE_READ command (see section 2.3.9) to query attribute *mStackStatus* (see section 2.3.12).

2.3.4. MSAP-STACK_STATE Service

The MSAP-STACK_STATE service informs the stack state at boot. MSAP-STACK_STATE service includes the following primitives:

- MSAP-STACK_STATE.indication
- MSAP-STACK_STATE.response (All response primitives have the same format, see section 2.5)

2.3.4.1. MSAP-STACK_STATE.indication

The MSAP-STACK_STATE.indication is issued by the stack when it has booted. The MSAP-STACK_STATE.indication frame is depicted in Figure 14.

Primitive ID	Frame ID	Payload length	Indication status	Status	CRC
1 octet	1 octet	1 octet	1 octet	1 octet	2 octets

Figure 14. MSAP-STACK_STATE.indication frame

The MSAP-STACK_STATE.indication frame fields are explained in Table 14.

Table 14. MSAP-STACK_STATE.indication frame fields

Field Name	Size	Valid Values	Description
IndicationStatus	1	0 or 1	0 = No other indications queued 1 = More indications queued
Status	1	0xxx xxxx (where x can be 0 or 1)	The stack status is indicated as a bit field with individual bits defined as follows: Bit 0 = 0: Stack running, see Note below Bit 0 = 1: Stack stopped Bit 1 = 0: Network address set Bit 1 = 1: Network address missing Bit 2 = 0: Node address set Bit 2 = 1: Node address missing Bit 3 = 0: Network channel set Bit 3 = 1: Network channel missing Bit 4 = 0: Role set Bit 4 = 1: Role missing Bit 5 = 0: Application configuration data valid Bit 5 = 1: Application configuration data missing (valid only on sink device) Bit 7: Reserved , where bit 0 is the least significant bit and bit 7 is the most significant bit.

Note: If the stack sends an MSAP-STACK_STATE.indication where the status bit 0 = 0, it means that the stack has auto-started. If the status bit 0 = 1, it means that auto-start is disabled. For more information on the auto-start feature, see section 2.3.12 (MSAP auto-start attribute).

2.3.5. MSAP-APP_CONFIG_DATA_WRITE Service

The MSAP-APP_CONFIG_DATA_WRITE service can be used for two things:

1. Configure application-specific parameters (application configuration data) to the application running in the nodes (via the network)

2. Configure transmission interval of the stack diagnostic data

The application configuration data is persistent global data for the whole network. The data format can be decided by the application. Application configuration data can be set by the sinks' application after which it is disseminated to the whole network and stored at every node. It can include e.g. application parameters, such as measurement interval. The service makes it possible to set the data, after which every new node joining the network receives the data from its neighbors without the need for end-to-end polling. Furthermore, new configurations can be set and updated to the network on the run.

The MSAP-APP_CONFIG_DATA_WRITE service includes the following primitives:

- MSAP-APP_CONFIG_DATA_WRITE.request
- MSAP-APP_CONFIG_DATA_WRITE.confirm

Note 1: The MSAP-APP_CONFIG_DATA_WRITE service can only be used in sink role.

Note 2: In a network including multiple sinks, the same configuration data should be set to all sinks so that it can be guaranteed to disseminate to every node.

Note 3: Application configuration data is stored in permanent memory similarly to the persistent attributes. To avoid memory wearing, do not write new values too often (e.g. more often than once per 30 minutes).

2.3.5.1. MSAP-APP_CONFIG_DATA_WRITE.request

The MSAP-APP_CONFIG_DATA_WRITE.request is issued by the application when it wants to set or change the network configuration data contents. The MSAP-APP_CONFIG_DATA_WRITE.request frame is depicted in Figure 15.

Primitive ID	Frame ID	Payload length	Sequence number	Diagnostic data interval	App config data	CRC
1 octet	1 octet	1 octet	1 octet	2 octets	X octets	2 octets

Figure 15. MSAP-APP_CONFIG_DATA_WRITE.request frame

The MSAP-APP_CONFIG_DATA_WRITE.request frame fields are explained in Table 15.

Table 15. MSAP-APP_CONFIG_DATA_WRITE.request frame fields

Field Name	Size	Valid Values	Description
SequenceNumber	1	0 – 254, default value is 0	Sequence number for filtering old and already received application configuration data packets at the nodes. The sequence number must be increment by 1 every time new configuration is written, i.e. new diagnostic data interval and/or new application configuration data is updated. See section 2.6 for details. A sequence number that is the current value of existing application configuration data is invalid. A value of 255 is invalid. Therefore, after value of 254, the next valid value is 0.

Field Name	Size	Valid Values	Description
DiagnosticDataInterval	2	0, 30, 60, 120, 300, 600, 1800, default value is 0	Diagnostic data transmission interval in seconds, i.e. how often the nodes on the network should send diagnostic PDUs. If the value is 0, diagnostic data transmission is disabled.
AppConfigData	X	Raw hex data, default value is filled with 0x00	Application configuration data. The format can be decided by the application. Size of the field is defined by CSAP attribute cAppConfigDataSize (see section 2.4.4.16)

Note: It is recommended that the configuration data is not written too often, as new configuration data is always written to the non-volatile memory of the sink and disseminated to the network. This can cause unnecessary wearing of the memory with devices that need to use the program memory to store persistent variables and unnecessary load to the network.

2.3.5.2. Reserved values in AppConfigData

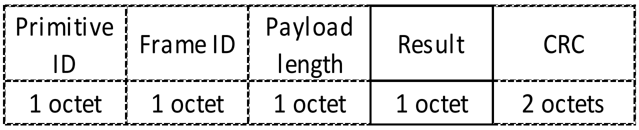
The first byte of the AppConfigData has some reserved values. Do not use these values as first byte of your AppConfigData.

Table 16. Reserved values of the first byte of AppConfigData

First byte value	Description
0x02	This value is reserved for configuring IPv6 functionality in the Wirepas network. If the network does not support IPv6 functionality, this value can be used freely.

2.3.5.3. MSAP-APP_CONFIG_DATA_WRITE.confirm

The MSAP-APP_CONFIG_DATA_WRITE.confirm is issued by the stack in response to the MSAP-APP_CONFIG_DATA_WRITE.request. The MSAP-APP_CONFIG_DATA_WRITE.confirm frame is depicted in Figure 16.



Primitive ID	Frame ID	Payload length	Result	CRC
1 octet	1 octet	1 octet	1 octet	2 octets

Figure 16. MSAP-APP_CONFIG_DATA_WRITE.confirm frame

The MSAP-APP_CONFIG_DATA_WRITE.confirm frame fields are explained in Table 17.

Table 17. MSAP-APP_CONFIG_DATA_WRITE.confirm frame fields

Field Name	Size	Valid Values	Description
Result	1	0 – 4	The return result of the corresponding MSAP-APP_CONFIG_DATA_WRITE.request. The different values are defined as follows: 0 = Success: New configuration written to sink's non-volatile memory and scheduled for transmission 1 = Failure: The node is not a sink 2 = Failure: Invalid DiagnosticDataInterval value 3 = Failure: Invalid SequenceNumber value 4 = Failure: Access denied (see section 2.4.4.21)

2.3.6. MSAP-APP_CONFIG_DATA_READ Service

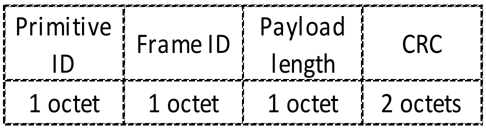
The MSAP-APP_CONFIG_DATA_READ service can be used to read the network-wide application configuration data. The MSAP-APP_CONFIG_DATA_READ service includes the following primitives:

- MSAP-APP_CONFIG_DATA_READ.request
- MSAP-APP_CONFIG_DATA_READ.confirm

For sinks, the service returns the configuration data last written to the stack using the MSAP-APP_CONFIG_DATA_WRITE service. For nodes, the service returns the configuration data that was last received from neighboring nodes.

2.3.6.1. MSAP-APP_CONFIG_DATA_READ.request

The MSAP-APP_CONFIG_DATA_READ.request is issued by the application when it wants to read the network configuration data contents. The MSAP-APP_CONFIG_DATA_READ.request frame is depicted in Figure 17.



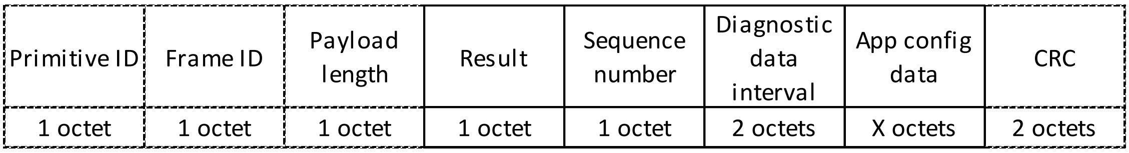
Primitive ID	Frame ID	Payload length	CRC
1 octet	1 octet	1 octet	2 octets

Figure 17. MSAP-APP_CONFIG_DATA_READ.request frame

The MSAP-APP_CONFIG_DATA_READ.request frame does not contain any payload.

2.3.6.2. MSAP-APP_CONFIG_DATA_READ.confirm

The MSAP-APP_CONFIG_DATA_READ.confirm is issued by the stack in response to the MSAP-APP_CONFIG_DATA_READ.request. The MSAP-APP_CONFIG_DATA_READ.confirm frame is depicted in Figure 18.



Primitive ID	Frame ID	Payload length	Result	Sequence number	Diagnostic data interval	App config data	CRC
1 octet	1 octet	1 octet	1 octet	1 octet	2 octets	X octets	2 octets

Figure 18. MSAP-APP_CONFIG_DATA_READ.confirm frame

The MSAP-APP_CONFIG_DATA_READ.confirm frame fields are explained in Table 18.

Table 18. MSAP-APP_CONFIG_DATA_READ.confirm frame fields

Field Name	Size	Valid Values	Description
Result	1	0 – 2	Return result for the corresponding MSAP-APP_CONFIG_DATA_READ.request. The different values are defined as follows: 0 = Success: Configuration received/set 1 = Failure: No configuration received/set 2 = Failure: Access denied (see section 2.4.4.21) When used with nodes, indicates whether configuration has been received from neighbors. When used with sinks, indicates whether configuration has already been set (by using MSAP-APP_CONFIG_DATA_WRITE service).
SequenceNumber	1	0 – 254	Sequence number for filtering old and already received application configuration data packets at the nodes. This parameter can be used by the application to decide if the configuration data has been updated. See section 2.6 for details. The returned value is never 255.
DiagnosticDataInterval	2	0, 30, 60, 120, 300, 600, 1800	Diagnostic data transmission interval in seconds, i.e. how often the stack should send diagnostic PDUs If the value is 0, diagnostic data transmission is disabled.
AppConfigData	X	-	Application configuration data. The format can be decided by the application. Size of the field is defined by CSAP attribute cAppConfigDataSize (see section 2.4.4.16)

2.3.7. MSAP-APP_CONFIG_DATA_RX Service

The MSAP-APP_CONFIG_DATA_RX service provides support for asynchronous sending of received configuration data to the application when new data is received from the network. The MSAP-APP_CONFIG_DATA_RX service includes the following primitives:

- MSAP-APP_CONFIG_DATA_RX.indication
- MSAP-APP_CONFIG_DATA_RX.response (All response primitives have the same format, see section 2.5)

Note: The MSAP-APP_CONFIG_DATA_RX service is available only in node role. With sinks the configuration is set with MSAP-APP_CONFIG_DATA_WRITE service as described above.

2.3.7.1. MSAP-APP_CONFIG_DATA_RX.indication

The MSAP-APP_CONFIG_DATA_RX.indication is issued by the stack when it receives new configuration data from its neighbors. The MSAP-APP_CONFIG_DATA_RX.indication frame is similar to MSAP-APP_CONFIG_DATA_read.confirm frame (see section 2.3.6.2). Only exception is that the *Result*-parameter is replaced with the *indicationStatus* field (see section 2.3.1.2).

Primitive ID	Frame ID	Payload length	Result	Sequence number	Diagnostic data interval	App config data	CRC
1 octet	1 octet	1 octet	1 octet	1 octet	2 octets	X octets	2 octets

Figure 19. MSAP-APP_CONFIG_DATA_RX.indication frame

2.3.8. MSAP-ATTRIBUTE_WRITE Service

The MSAP-ATTRIBUTE_WRITE service can be used by the application to write run-time management attributes of the stack. The MSAP-ATTRIBUTE_WRITE service includes the following primitives:

- MSAP-ATTRIBUTE_WRITE.request
- MSAP-ATTRIBUTE_WRITE.confirm

The MSAP attributes are specified in section 2.3.14.

2.3.8.1. MSAP-ATTRIBUTE_WRITE.request

The MSAP-ATTRIBUTE_WRITE.request is issued by the application when it wants to set or change the MSAP attributes. The MSAP-ATTRIBUTE_WRITE.request frame is depicted in Figure 20.

Primitive ID	Frame ID	Payload length	Attribute ID	Attribute Length	Attribute value	CRC
1 octet	1 octet	1 octet	2 octets	1 octet	1-16 octets	2 octets

Figure 20. MSAP-ATTRIBUTE_WRITE.request frame

The MSAP-ATTRIBUTE_WRITE.request frame fields are explained in Table 19.

Table 19. MSAP-ATTRIBUTE_WRITE.request frame fields

Field Name	Size	Valid Values	Description
AttributeID	2	Depends on the attribute. See section 2.3.14	The ID of the attribute that is written
AttributeLength	1		The length (in octets) of the attribute that is written
AttributeValue	1 – 16		The value that is written to the attribute specified by the set attribute ID

2.3.8.2. MSAP-ATTRIBUTE_WRITE.confirm

The MSAP-ATTRIBUTE_WRITE.confirm is issued by the stack in response to the MSAP-ATTRIBUTE_WRITE.request. The MSAP-ATTRIBUTE_WRITE.confirm frame is depicted in Figure 21.

Primitive ID	Frame ID	Payload length	Result	CRC
1 octet	1 octet	1 octet	1 octet	2 octets

Figure 21. MSAP-ATTRIBUTE_WRITE.confirm frame

The MSAP-ATTRIBUTE_WRITE.confirm frame fields are explained in

Table 20.

Table 20. MSAP-ATTRIBUTE_WRITE.confirm frame fields

Field Name	Size	Valid Values	Description
Result	1	0 – 6	The return result of the corresponding MSAP-ATTRIBUTE_WRITE.request. The different values are defined as follows: 0 = Success 1 = Failure: Unsupported attribute ID 2 = Failure: Stack in invalid state to write attribute 3 = Failure: Invalid attribute length 4 = Failure: Invalid attribute value 5 = Reserved 6 = Failure: Access denied (e.g. attribute read prevented by feature lock bits)

2.3.9. MSAP-ATTRIBUTE_READ Service

The MSAP-ATTRIBUTE_READ service can be used by the application layer to read run-time management attributes from the stack. The MSAP-ATTRIBUTE_READ service includes the following primitives:

- MSAP-ATTRIBUTE_READ.request
- MSAP-ATTRIBUTE_READ.confirm

2.3.9.1. MSAP-ATTRIBUTE_READ.request

The MSAP-ATTRIBUTE_READ.request is issued by the application when it wants to read the MSAP attributes. The MSAP-ATTRIBUTE_READ.request frame is depicted in Figure 22.

Primitive ID	Frame ID	Payload length	Attribute ID	CRC
1 octet	1 octet	1 octet	2 octets	2 octets

Figure 22. MSAP-ATTRIBUTE_READ.request frame

The MSAP-ATTRIBUTE_READ.request frame fields are explained in Table 21.

Table 21. MSAP-ATTRIBUTE_READ.request frame fields

Field Name	Size	Valid Values	Description
AttributeID	2	Depends on the attribute. See section 2.3.14	The ID of the attribute that is read

2.3.9.2. MSAP-ATTRIBUTE_READ.confirm

The MSAP- ATTRIBUTE_READ.confirm is issued by the stack in response to the MSAP-ATTRIBUTE_READ.request. The MSAP-ATTRIBUTE_READ.confirm frame is depicted in Figure 23.

Primitive ID	Frame ID	Payload length	Result	Attribute ID	Attribute length	Attribute value	CRC
1 octet	1 octet	1 octet	1 octet	2 octets	1 octet	1-16 octets	2 octets

Figure 23. MSAP-ATTRIBUTE_READ.confirm frame

The MSAP-ATTRIBUTE_READ.confirm frame fields are explained in Table 22.

Table 22. MSAP-ATTRIBUTE_READ.confirm frame fields

Field Name	Size	Valid Values	Description
Result	1	0 – 6	The return result of the corresponding MSAP-ATTRIBUTE_READ.request. The different values are defined as follows: 0 = Success 1 = Failure: Unsupported attribute ID 2 = Failure: Stack in invalid state to read attribute 4 = Failure: Invalid attribute value or attribute value not yet set 5 = Failure: Write-only attribute (e.g. the encryption and authentication keys) 6 = Failure: Access denied (e.g. attribute read prevented by feature lock bits)
AttributeID	2	Depends on the attribute. See section 2.3.14.	The ID of the attribute that is read
AttributeLength	1		The length (in octets) of the attribute that is read
AttributeValue	1 – 16		The value of the read attribute specified by the set attribute ID. This value of the attribute is only present if *Result* is 0 (Success).

2.3.10. MSAP-GET_NBORS Service

This service can be used to tell the status of a node’s neighbors. This information may be used for various purposes, for example to estimate where a node is located.

2.3.10.1. MSAP-GET_NBORS.request

MSAP-GET_NBORS.request is issued by the application layer to query information about neighboring nodes. The MSAP-GET_NBORS.request frame is depicted in Figure 24. It contains no payload.

Primitive ID	Frame ID	Payload length	CRC
1 octet	1 octet	1 octet	2 octets

Figure 24. MSAP-GET_NBORS.request frame

2.3.10.2. MSAP-GET_NBORS.confirm

MSAP-GET_NBORS.confirm is issued by the stack as a response to the MSAP-GET_NBORS.request. The MSAP-GET_NBORS.confirm frame is depicted in Figure 25 and Figure 26.

Information for a maximum of eight neighbors is contained within one MSAP-GET_NBORS.confirm frame. The frame size does not change. If number of neighbors is less than eight, remaining data in the frame is undefined. If access is denied (see section 2.4.4.21) a block of zeros is returned.

Primitive ID	Frame ID	Payload length	Number of neighbors	Neighbor info	CRC
1 octet	1 octet	1 octet	1 octet	13 octets	2 octets
				x 8 times	

Figure 25. MSAP-GET_NBORS.confirm frame

The neighbor info frame contains information for a maximum of eight neighbors. This information is depicted in Figure 26.

Neighbor address	Link reliability	Normalized RSSI	Cost	Channel	Neighbor type	TX Power	RX Power	Last update
4 octets	1 octet	1 octet	1 octet	1 octet	1 octet	1 octet	1 octet	2 octets

Figure 26. MSAP-GET_NBORS.confirm neighbor info

The MSAP-GET_NBORS.confirm frame fields are explained in Table 23.

Table 23. MSAP-GET_NBORS.confirm frame fields

Field Name	Size	Valid Values	Description
NumberOfNeighbors	1	0 – 8	Number of neighbors' information returned. 0 if there are no neighbors.
NeighborAddress	4	0 – 4294967295	Address of the neighbor node
LinkReliability	1	0 – 255	Link reliability to the neighboring node. Scaled so that 0 = 0 %, 255 = 100 %
NormalizedRSSI	1	0 – 255	Received signal strength, compensated with transmission power. Larger value means better the signal. 0: No signal 1: Signal heard barely >50: Good signal
Cost	1	1 – 255	Route cost to the sink via this neighbor. Value 255 indicates that a neighbor has no route to a sink.
Channel	1	1 – CSAP attribute *cChannelLimits*	Radio channel used by the neighbor
NeighborType	1	0 – 2	Type of neighbor 0: Neighbor is next hop cluster, i.e. used as a route to sink 1: Neighbor is a member of this node 2: Neighbor is a cluster from network scan
TxPower	1	0 – X	Power level used for transmission 0: Lowest power X: Highest power (depending on the stack profile)
RxPower	1	0 – X	Received power level 0: Lowest power X: Highest power (depending on the stack profile)
LastUpdate	2	0 – 65535	Amount of seconds since these values were last updated

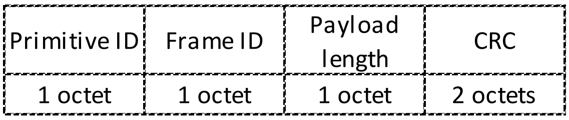
2.3.11. MSAP-SCAN_NBORS Service

This service can be used by the application to get fresh information about neighbors. Application can trigger to measurement all neighbors and once the measurement is done, application is informed it over API. The MSAP-SCAN_NBORS service includes the following primitives:

- MSAP-SCAN_NBORS.request
- MSAP-SCAN_NBORS.confirm
- MSAP-SCAN_NBORS.indication
- MSAP-SCAN_NBORS.response (All response primitives have the same format, see section 2.5)

2.3.11.1. MSAP-SCAN_NBORS.request

MSAP-SCAN_NBORS.request is issued by the application when it starts to measurement all neighbors. The MSAP-SCAN_NBORS.request frame is depicted in Figure 24. It contains no payload.

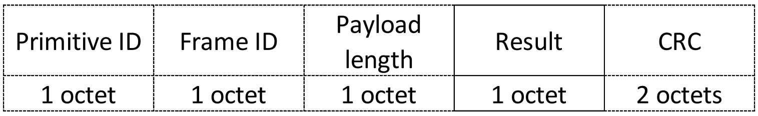


Primitive ID	Frame ID	Payload length	CRC
1 octet	1 octet	1 octet	2 octets

Figure 27. MSAP-SCAN_NBORS.request frame

2.3.11.2. MSAP-SCAN_NBORS.confirm

This confirm tells result which is always success. After application has asked to scan neighbors so stack code use signal to handle it.



Primitive ID	Frame ID	Payload length	Result	CRC
1 octet	1 octet	1 octet	1 octet	2 octets

Figure 28. MSAP-SCAN_NBORS.confirm frame

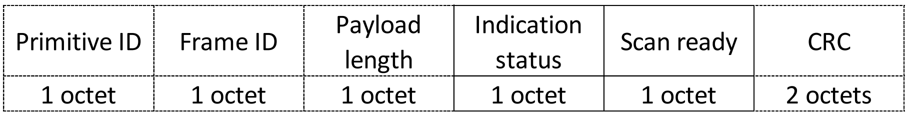
The MSAP-SCAN_NBORS.confirm frame fields are explained in following table.

Table 24. MSAP_SCAN_NBORS.confirm frame fields

Field Name	Size	Valid Values	Description
Result	1	0 – 2	The return result of the corresponding MSAP-SCAN_NBORS.confirm. 0 = Success 1 = Failure: Stack in invalid state, i.e. not running 2 = Failure: Access denied (see section 2.4.4.21)

2.3.11.3. MSAP-SCAN_NBORS.indication

The MSAP-SCAN_NBORS.indication is issued by the stack as an asynchronous reply for the MSAP-SCAN_NBORS.request after to scan neighbors is finished. The MSAP-SCAN_NBORS.indication frame is depicted in Figure 29.



Primitive ID	Frame ID	Payload length	Indication status	Scan ready	CRC
1 octet	1 octet	1 octet	1 octet	1 octet	2 octets

Figure 29. MSAP-SCAN_NBORS.indication frame

The MSAP-SCAN_NBORS.indication frame fields are explained in following table.

Table 25. MSAP-SCAN_NBORS.indication frame fields

Field Name	Size	Valid Values	Description
IndicationStatus	1	0 or 1	0 = No other indications queued 1 = More indications queued
ScanReady	1	1	1 = Scan is done

2.3.12. MSAP-SINK_COST Service

This service can be used to inform the sink that the backend communication has problems. In order to keep the entire network operational, other nodes can be forced to use other sinks with working backend communication.

2.3.12.1. MSAP-SINK_COST_WRITE.request

This command shall set the sink cost to new value. The higher the cost value, more this sink shall be avoided.

Primitive ID	Frame ID	Payload length	Cost	CRC
1 octet	1 octet	1 octet	1 octet	2 octets

Figure 30: MSAP-SINK_COST_WRITE.request frame

The MSAP-ATTRIBUTE_WRITE.request frame fields are explained in following table.

Table 26. MSAP-ATTRIBUTE_WRITE.confirm frame fields

Field Name	Size	Valid Values	Description
Cost	1	0 – 254	Value of 0 means that connection is good and no additional penalty is sent to sink usage. Value of 254 includes maximum penalty

2.3.12.2. MSAP-SINK_COST_WRITE.confirm

This response tells whether writing of the penalty is successful or not.

Primitive ID	Frame ID	Payload length	Result	CRC
1 octet	1 octet	1 octet	1 octet	2 octets

Figure 31: MSAP-SINK_COST_WRITE.confirm frame

The MSAP-SINK_COST_WRITE.confirm frame fields are explained in following table.

Table 27. MSAP SINK_COST_WRITE.confirm frame fields

Field Name	Size	Valid Values	Description
Result	1	0 – 2	The return result of the corresponding MSAP-ATTRIBUTE_WRITE.request. The different values are defined as follows: 0 = Success 1 = Failure: Device is not a sink 2 = Failure: Access denied (see section 2.4.4.21)

2.3.12.3. MSAP-SINK_COST_READ.request

This command is used to query the currently set additional penalty for the sink usage.

Primitive ID	Frame ID	Payload length	CRC
1 octet	1 octet	1 octet	2 octets

Figure 32: MSAP-SINK_COST_READ.request frame

2.3.12.4. MSAP-SINK_COST_READ.confirm

This response tells currently set additional penalty for the sink usage.

Primitive ID	Frame ID	Payload length	Result	Cost	CRC
1 octet	1 octet	1 octet	1 octet	1 octet	2 octets

Figure 33: MSAP-SINK_COST_READ.confirm frame

The MSAP-SINK_COST_READ.confirm fields are explained in the following table:

Table 28. MSAP- SINK_COST_READ.confirm frame fields

Field Name	**Size**	**Valid Values**	**Description**
Result	1	0 – 2	The return result of the corresponding MSAP-ATTRIBUTE_WRITE.request. The different values are defined as follows: 0 = Success 1 = Failure: Device is not a sink 2 = Failure: Access denied (see section 2.4.4.21)
Cost	1	0 – 254	Additional penalty set for the sink

2.3.13. MSAP-SCRATCHPAD Services

Wirepas Mesh stack support a feature called Over-The-Air-Programming (OTAP). Nodes reserve any unused non-volatile memory as storage for large bulk data. This area is called the OTAP scratchpad. OTAP scratchpad can be used to distribute new firmware, applications and other large pieces of data to all nodes on the network. Nodes keep track of the scratchpad sequence numbers of their neighbors, and coordinate the distribution of the most recent OTAP scratchpad to all nodes.

Any node can be used to introduce a new OTAP scratchpad to the network, but the stack must be in the stopped state while writing the scratchpad data. After the stack is started, the OTAP transfer will begin (unless disabled for this node).

Note: It is recommended that the scratchpad data is not rewritten too often, as new data is always written to the non-volatile memory of the sink and distributed to all nodes on the network. This can cause unnecessary wearing of the non-volatile memory and unnecessary load to the network.

2.3.13.1. MSAP-SCRATCHPAD_START.request

The MSAP-SCRATCHPAD_START.request is issued by the application when it wants to clear and rewrite the Scratchpad contents of this node. Any previous scratchpad contents are erased.

Note:

- This request is only valid when the stack is in the stopped state
- The length of the scratchpad in bytes must be divisible by 16

The MSAP-SCRATCHPAD_START.request frame is depicted in Figure 34.

Primitive ID	Frame ID	Payload length	Scratchpad length in bytes	Scratchpad sequence number	CRC
1 octet	1 octet	1 octet	4 octets	1 octet	2 octets

Figure 34. MSAP-SCRATCHPAD_START.request frame

The MSAP-SCRATCHPAD_START.request frame fields are explained in Table 29.

Table 29. MSAP-SCRATCHPAD_START.request frame fields

Field Name	Size	Valid Values	Description
ScratchpadLengthInBytes	4	96 –	Total number of bytes of OTAP scratchpad data The length must be divisible by 16, or the request will fail.
ScratchpadSequenceNumber	1	0 – 255	Sequence number for filtering old scratchpad contents at the nodes The sequence number must be increment by 1 every time a new OTAP scratchpad is written. See section 2.6 for details. The following sequence numbers are considered special: • The sequence number must be different to the sequence number of the scratchpad already present in the node • A value of 255 is means that any scratchpad from the network will override this scratchpad • A value of 0 disables OTAP for this node

2.3.13.2. MSAP-SCRATCHPAD_START.confirm

The MSAP-SCRATCHPAD_START.confirm is issued by the stack in response to the MSAP-SCRATCHPAD_START.request. The MSAP-SCRATCHPAD_START.confirm frame is depicted in Figure 35.

Primitive ID	Frame ID	Payload length	Result	CRC
1 octet	1 octet	1 octet	1 octet	2 octets

Figure 35. MSAP-SCRATCHPAD_START.confirm frame

The MSAP-SCRATCHPAD_START.confirm frame fields are explained in Table 30.

Table 30. MSAP-SCRATCHPAD_START.confirm frame fields

Field Name	Size	Valid Values	Description
Result	1	0 – 4	The return result of the corresponding MSAP-SCRATCHPAD_START.request. The different values are defined as follows: 0 = Success: Scratchpad has been erased and the node is waiting for new data to be written 1 = Failure: Stack in invalid state, i.e. not stopped 2 = Failure: Invalid ScratchPadLengthInBytes value, e.g. too big or not divisible by 16 3 = Reserved 4 = Failure: Access denied (see section 2.4.4.21)

2.3.13.3. MSAP-SCRATCHPAD_BLOCK.request

Blocks of data for the OTAP scratchpad are written with MSAP-SCRATCHPAD_BLOCK.request. Number of bytes of data to write can vary, with the following limitations:

- Number of bytes must be a multiple of four
- There may not be gaps in the blocks of data written and data from previously written blocks may not be overwritten
- The total number of bytes written with consecutive MSAP block requests must be equal to the number of bytes indicated in the preceding MSAP-SCRATCHPAD_START.request

MSAP-SCRATCHPAD_BLOCK.request frame is depicted in Figure 36.

Primitive ID	Frame ID	Payload length	Start address	Number of bytes	Bytes	CRC
1 octet	1 octet	1 octet	4 octets	1 octet	1-112 octets	2 octets

Figure 36. MSAP-SCRATCHPAD_BLOCK.request frame

The MSAP-SCRATCHPAD_BLOCK.request frame fields are explained in Table 31.

Table 31. MSAP-SCRATCHPAD_BLOCK.request frame fields

Field Name	**Size**	**Valid Values**	**Description**
StartAddress	4	0 –	Start address of scratchpad data Overlapping previous data or leaving gaps is not permitted.
NumberOfBytes	1	1 – 112	Number of bytes of scratchpad data Must be a multiple of four bytes.
Bytes	1 - 112	-	Bytes of scratchpad data

2.3.13.4. MSAP-SCRATCHPAD_BLOCK.confirm

The MSAP-SCRATCHPAD_BLOCK.confirm is issued by the stack in response to the MSAP-SCRATCHPAD_BLOCK.request. The MSAP-SCRATCHPAD_BLOCK.confirm frame is depicted in Figure 37.

Primitive ID	Frame ID	Payload length	Result	CRC
1 octet	1 octet	1 octet	1 octet	2 octets

Figure 37. MSAP-SCRATCHPAD_BLOCK.confirm frame

The MSAP-SCRATCHPAD_BLOCK.confirm frame fields are explained in Table 32.

Table 32. MSAP-SCRATCHPAD_BLOCK.confirm frame fields

Field Name	Size	Valid Values	Description
Result	1	0 – 7	The return result of the corresponding MSAP-SCRATCHPAD_BLOCK.request. The different values are defined as follows: 0 = Success: Block was accepted 1 = Success: All data received and seems to be OK 2 = Failure: All data received but error in data 3 = Failure: Stack in invalid state, i.e. not stopped 4 = Failure: No scratchpad start request was given 5 = Failure: Start address is invalid 6 = Failure: Number of bytes is invalid 7 = Failure: Does not seem to be a valid scratchpad

Note: Any other result except 0 (block was accepted) means that writing the scratchpad has been terminated and another MSAP-SCRATCHPAD_START.request must be issued before another MSAP-SCRATCHPAD_BLOCK.request can be carried out.

2.3.13.5. MSAP-SCRATCHPAD_STATUS.request

MSAP-SCRATCHPAD_STATUS.request is issued by the application layer to query information about the OTAP scratchpad present in the node, as well as information about the scratchpad that produced the currently running stack firmware. The MSAP-SCRATCHPAD_STATUS.request frame is depicted in Figure 38.

Primitive ID	Frame ID	Payload length	CRC
1 octet	1 octet	1 octet	2 octets

Figure 38. MSAP-SCRATCHPAD_STATUS.request frame

The MSAP-SCRATCHPAD_STATUS.request frame does not contain any payload.

2.3.13.6. MSAP-SCRATCHPAD_STATUS.confirm

The MSAP-SCRATCHPAD_STATUS.confirm is issued by the stack in response to the MSAP-SCRATCHPAD_STATUS.request. There are several kinds of information in the status confirmation:

- Information about the OTAP scratchpad currently present in the node: its length, CRC, sequence number, etc.
- Information about the scratchpad that produced the currently running firmware: its length, CRC, sequence number, etc.
- Information about the currently running stack firmware: version numbers

By keeping track of several identifying pieces of information about the OTAP scratchpad that produced the currently running stack firmware, it is possible to unambiguously determine how the stack firmware ended up in the node.

If access is denied (see section 2.4.4.21) a block of zeros is returned. The MSAP-SCRATCHPAD_STATUS.confirm frame is depicted in Figure 39.

Primitive ID	Frame ID	Payload length	Scratchpad length in bytes	Scratchpad CRC	Scratchpad sequence number	Scratchpad type	Scratchpad status
1 octet	1 octet	1 octet	4 octets	2 octets	1 octet	1 octet	1 octet

…

Processed scratchpad length in bytes	Processed scratchpad CRC	Processed scratchpad sequence number	Firmware memory area ID	Firmware major version	Firmware minor version	Firmware maintenance version	Firmware development version	CRC
4 octets	2 octets	1 octet	4 octets	1 octet	1 octet	1 octet	1 octet	2 octets

Figure 39. MSAP-SCRATCHPAD_STATUS.confirm frame

The MSAP-SCRATCHPAD_STATUS.confirm frame fields are explained in Table 33.

Table 33. MSAP-SCRATCHPAD_STATUS.confirm frame fields

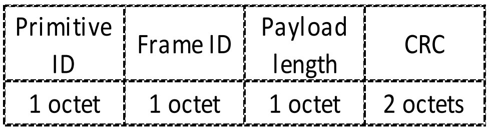
Field Name	**Size**	**Valid Values**	**Description**
ScratchpadLengthInBytes	4	0 or 96 –	Length of the OTAP scratchpad present in the node Length is 0 if there is no scratchpad present in the node
ScratchpadCrc	2	0 – 65535	CRC of the OTAP scratchpad present in the node CRC is 0 if there is no scratchpad present in the node
ScratchpadSequenceNumber	1	0 – 255	Sequence number of the OTAP scratchpad present in the node Sequence number is 0 if there is no scratchpad present in the node
ScratchpadType	1	0 – 2	Type of the OTAP scratchpad present in the node. Type can be: 0 = Blank: No valid scratchpad is present 1 = Present: A valid scratchpad is present, but has not been marked to be processed 2 = Process: A valid scratchpad is present and has been marked to be processed

Field Name	**Size**	**Valid Values**	**Description**
ScratchpadStatus	1	0 – 255	Status of the OTAP scratchpad present in the node: 255 = New: Bootloader has not yet processed the scratchpad 0 = Success: Bootloader has processed the scratchpad successfully 1 – 254 = Error: Bootloader encountered an error while processing the scratchpad Status is 0 also if there is no scratchpad present in the node
ProcessedScratchpadLengthInBytes	4	0 or 96 –	Length of the OTAP scratchpad that produced the firmware currently running on the node
ProcessedScratchpadCrc	2	0 – 65535	CRC of the OTAP scratchpad that produced the firmware currently running on the node
ProcessedScratchpadSequenceNumber	1	0 – 255	Sequence number of the OTAP scratchpad that produced the firmware currently running on the node
FirmwareMemoryAreaId	4	Any	Memory area ID of the file in the OTAP scratchpad that produced the firmware currently running on the node OTAP scratchpad may contain multiple firmware images. This value can be used to determine which one the bootloader picked.
FirmwareMajorVersion	1	0 – 255	Major version number of currently running firmware
FirmwareMinorVersion	1	0 – 255	Minor version number of currently running firmware
FirmwareMaintenanceVersion	1	0 – 255	Maintenance version number of currently running firmware
FirmwareDevelopmentVersion	1	0 – 255	Development version number of currently running firmware

2.3.13.7. MSAP-SCRATCHPAD_UPDATE.request

The application issues MSAP-SCRATCHPAD_UPDATE.request to mark the OTAP scratchpad for processing by the bootloader. The bootloader will process the scratchpad contents on next reboot. See section 2.3.3.1 on how to reboot the node. Note, that this request is only valid when the stack is in the stopped state.

MSAP-SCRATCHPAD_UPDATE.request frame is depicted in Figure 40.



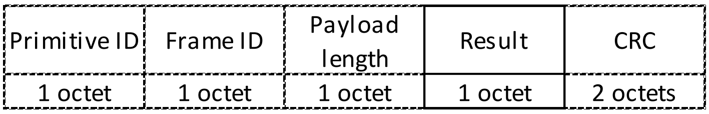
Primitive ID	Frame ID	Payload length	CRC
1 octet	1 octet	1 octet	2 octets

Figure 40. MSAP-SCRATCHPAD_UPDATE.request frame

The MSAP-SCRATCHPAD_UPDATE.request frame does not contain any payload.

2.3.13.8. MSAP-SCRATCHPAD_UPDATE.confirm

The MSAP-SCRATCHPAD_UPDATE.confirm is issued by the stack in response to the MSAP-SCRATCHPAD_UPDATE.request. The MSAP-SCRATCHPAD_UPDATE.confirm frame is depicted in Figure 41.



Primitive ID	Frame ID	Payload length	Result	CRC
1 octet	1 octet	1 octet	1 octet	2 octets

Figure 41. MSAP-SCRATCHPAD_UPDATE.confirm frame

The MSAP-SCRATCHPAD_UPDATE.confirm frame fields are explained in Table 34.

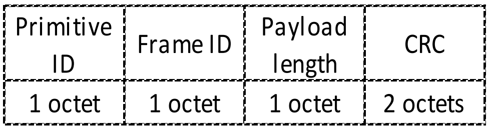
Table 34. MSAP-SCRATCHPAD_UPDATE.confirm frame fields

Field Name	Size	Valid Values	Description
Result	1	0 – 3	The return result of the corresponding MSAP-SCRATCHPAD_UPDATE.request. The different values are defined as follows: 0 = Success: Bootloader may process the scratchpad 1 = Failure: Stack in invalid state, i.e. not stopped 2 = Failure: No valid OTAP scratchpad present 3 = Failure: Access denied (see section 2.4.4.21)

2.3.13.9. MSAP-SCRATCHPAD_CLEAR.request

The MSAP-SCRATCHPAD_CLEAR.request is issued by the application when it wants to erase the OTAP scratchpad. Note, that this request is only valid when the stack is in the stopped state.

MSAP-SCRATCHPAD_CLEAR.request frame is depicted in Figure 42.



Primitive ID	Frame ID	Payload length	CRC
1 octet	1 octet	1 octet	2 octets

Figure 42. MSAP-SCRATCHPAD_CLEAR.request frame

The MSAP-SCRATCHPAD_CLEAR.request frame does not contain any payload.

2.3.13.10. MSAP-SCRATCHPAD_CLEAR.confirm

The MSAP-SCRATCHPAD_CLEAR.confirm is issued by the stack in response to the MSAP-SCRATCHPAD_CLEAR.request. The MSAP-SCRATCHPAD_CLEAR.confirm frame is depicted in Figure 43.

Primitive ID	Frame ID	Payload length	Result	CRC
1 octet	1 octet	1 octet	1 octet	2 octets

Figure 43. MSAP-SCRATCHPAD_CLEAR.confirm frame

The MSAP-SCRATCHPAD_CLEAR.confirm frame fields are explained in Table 35.

Table 35. MSAP-SCRATCHPAD_CLEAR.confirm frame fields

Field Name	Size	Valid Values	Description
Result	1	0 – 1	The return result of the corresponding MSAP-SCRATCHPAD_CLEAR.request. The different values are defined as follows: 0 = Success: Scratchpad has been erased 1 = Failure: Stack in invalid state, i.e. not stopped 2 = Failure: Access denied (see section 2.4.4.21)

2.3.14. MSAP-NON-ROUTER LONG SLEEP (NRLS) Service

The Non-Router Long Sleep (NRLS) is a service used to sleep Wirepas Mesh stack for time periods. Once waking-up from the sleep, Wirepas Mesh stack wakes up from the sleep without system reset. During Wirepas Mesh stack sleep the NRLS services over Dual-MCU API are available. Before entering to NRLS sleep, Wirepas Mesh stack needs to be running.

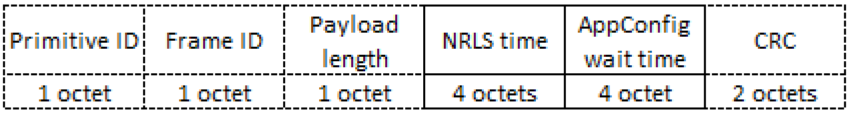
In order to use NRLS servicethe cNodeRole (as defined in Table 51) needs to be configured to stack as 0x03 (non router node). Any other node role settings do not allow to use NRLS functionality..

The MSAP-NON-ROUTER LONG SLEEP service includes following primitives:

- MSAP-NRLS.request
- MSAP-NRLS.confirm
- MSAP-NRLS_STOP.request
- MSAP-NRLS_STOP.confirm
- MSAP-NRLS_STATE_GET.request
- MSAP-NRLS_STATE_GET.response
- MSAP-NRLS_GOTOSLEEP_INFO.request
- MSAP-NRLS_GOTOSLEEP_INFO.response

2.3.14.1. MSAP-NRLS.request

This command shall start the Wirepas Mesh stack sleep for defined time.



Primitive ID	Frame ID	Payload length	NRLS time	AppConfig wait time	CRC
1 octet	1 octet	1 octet	4 octets	4 octet	2 octets

Figure 44: MSAP-NRLS.request frame

The MSAP-NRLS.request frame field is explained in following table.

Table 36: MSAP-NRLS.request frame fields

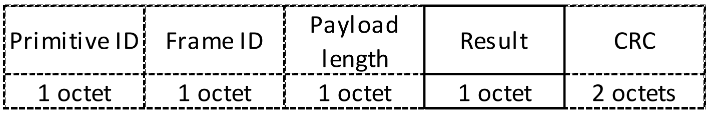
Field Name	Size	Valid Values	Description
NRLS time	4	0 to 0x93A80	0 = Starts Wirepas Mesh stack sleep for maximum value of 0x93A80 seconds (7 days) Other values = sleep time in seconds
Appconfig wait time	4	0x04 to 0x258	0 = App config is not awaited before going to sleep Other values = Time used to wait for app config data is received from network before going to sleep.

Wirepas Mesh stack sleep time is expressed in seconds. Sleep time starts when node gets disconnected from Mesh network. To disconnect from the network, some time is needed for signaling before disconnection is completed (the NRLS time does not include this time needed for signaling before going to sleep). Signaling before actual stack sleep start might take time up to 30 seconds or more depending of used radio.

If the time when waking up from NRLS sleep and going back to NRLS sleep is very short and app config is used to signal to the network e.g. overhaul state, good practice is to have Appconfig wait time long enough (minimum 4 seconds) to make sure that app config data is received before going to NRLS sleep (see more information in [1]).

2.3.14.2. MSAP-NRLS.confirm

The return result of the corresponding MSAP-NRLS.request is received in MSAP-NRLS.confirm message.



Primitive ID	Frame ID	Payload length	Result	CRC
1 octet	1 octet	1 octet	1 octet	2 octets

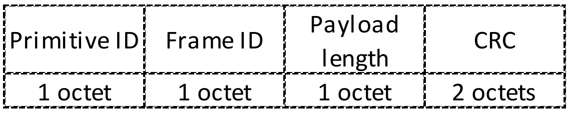
Figure 45: MSAP-NRLS.confirm frame

Table 37: MSAP-NRLS.confirm frame fields

Field Name	Size	Valid Values	Description
Result	1	0,1,2,5,6	0 = Success: NRLS started 1 = Failure: Invalid stack state 2 = Failure: Invalid stack role 5 = Failure: Invalid value 6 = Failure: Access denied (see section 2.4.4.21)

2.3.14.3. MSAP-NRLS_STOP.request

This command shall wakeup the Wirepas Mesh stack from NRLS sleep.

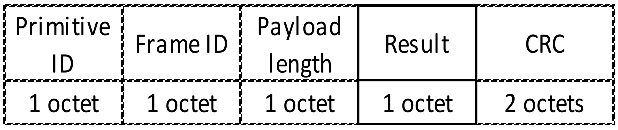


Primitive ID	Frame ID	Payload length	CRC
1 octet	1 octet	1 octet	2 octets

Figure 46:MSAP-NRLS_STOP.request

2.3.14.4. MSAP-NRLS_STOP.confirm

The MSAP-NRLS_STOP.confirm issued by the stack as a response to the MSAP-NRLS_STOP.request. The MSAP-NRLS_STOP.confirm frame is depicted in Figure 52.



Primitive ID	Frame ID	Payload length	Result	CRC
1 octet	1 octet	1 octet	1 octet	2 octets

Figure 47: MSAP-NRLS_STOP.confirm frame

The MSAP-NRLS_STOP.confirm frame fields are explained in following table.

Table 38: MSAP-NRLS_STOP.confirm frame fields

Field Name	Size	Valid Values	Description
Result	1	0	The return result of the corresponding MSAP-NRLS_STOP.request. The different values are defined as follows: 0 = Success: NRLS is stopped and stack is started 1 = Failure: Stack is not in sleep state (stopped) and NRLS stop cannot be done 6 = Failure: Access denied (see section 2.4.4.21)

2.3.14.5. MSAP-NRLS_STATE_GET.request

This command shall query NRLS state from the Wirepas Mesh stack.

Primitive ID	Frame ID	Payload length	CRC
1 octet	1 octet	1 octet	2 octets

Figure 48: MSAP-NRLS_STATE_GET.request

2.3.14.6. MSAP-NRLS_STATE_GET.response

The MSAP-NRLS_STATE_GET.response issued by the stack as a response to the MSAP-NRLS_STATE_GET.request.

Primitive ID	Frame ID	Payload length	State	Remain sleep time	CRC
1 octet	1 octet	5 octet	1 octet	4 octet	2 octets

Figure 49: MSAP-NRLS_SATE_GET.response

The MSAP-NRLS_STATE_GET.response frame fields are explained in following table.

Table 39: MSAP-NRLS_STATE_GET.response frame fields

Field Name	Size	Valid Values	Description
State	1	1,2	The return result of the corresponding MSAP-NRLS_STATE_GET.request. The different values are defined as follows: 1 = NRLS is active 2 = NRLS is not active
Remain sleep time	4	0-0x93A80	Remaining Wirepas Mesh stack sleep time in seconds. (Time is updated every 3 seconds)

2.3.14.7. MSAP-NRLS_GOTOSLEEP_INFO.request

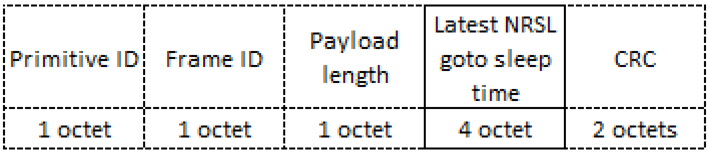
Request time in seconds which was used in previous NRLS sleep request starting from application NRLS sleep request until stack enters to NRLS sleep. Returned time in MSAP-NRLS_GOTOSLEEP_INFO.response includes total time used including application callbacks during that period.

Primitive ID	Frame ID	Payload length	CRC
1 octet	1 octet	1 octet	2 octets

Figure 50: MSAP-NRLS_GOTOSLEEP_INFO.request

2.3.14.8. MSAP- NRLS_GOTOSLEEP_INFO.response

The MSAP-NRLS_GOTOSLEEP_INFO.response issued by the stack as a response to the MSAP-NRLS_GOTOSLEEP.request.



Primitive ID	Frame ID	Payload length	Latest NRSL goto sleep time	CRC
1 octet	1 octet	1 octet	4 octet	2 octets

Figure 51: MSAP-NRLS_GOTOSLEEP_INFO.response

The MSAP-NRLS_GOTOSLEEP_INFO.response frame fields are explained in following table.

Table 40: MSAP-NRLS_GOTOSLEEP_INFO.response frame fields

Field Name	Size	Valid Values	Description
LatestNRLS goto sleep time	4	0 to 0x93A80	Time in seconds which was used in previous NRLS sleep request starting from application NRLS sleep request until stack enters to NRLS sleep. Time is total time used including application callbacks during that period.

2.3.15. MSAP-MAX_MESSAGE_QUEUING Service

The maximum message queuing services are used to change or read the current value of the time for how long the message is hold in the node's queue before it is discarded. Queuing time can be changed to normal and high priority messages.

Select queuing time carefully, too short value might cause unnecessary message drops and too big value filling up message queues. For consistent performance it is recommended to use the same queuing time in the whole network.

Minimum queuing time shall be bigger than access cycle interval in TDMA networks. It is recommended to use multiples of access cycle interval (+ extra) to give time for message repetitions, higher priority messages taking over the access slot etc. Access cycle is not limiting the minimum value in CSMA-CA networks.

Precision of the time when message is discarded depends on the checking interval of message's age. Interval is 1s in CSMA-CA and 15s for energy saving reasons in TDMA networks i.e. precision is 1s or 15s depending on used channel access method.

The MSAP-MAX_MESSAGE_QUEUING service includes following primitives:

- MSAP-MAX_QUEUE_TIME_WRITE.request
- MSAP-MAX_QUEUE_TIME_WRITE.confirm
- MSAP-MAX_QUEUE_TIME_READ.request
- MSAP-MAX_QUEUE_TIME_READ.confirm

2.3.15.1. MSAP-MAX_QUEUE_TIME_WRITE.request

This request is issued by the application layer to change the maximum queuing time for messages. The request frame is depicted in Figure 52.

Primitive ID	Frame ID	Payload length	Priority	Time	CRC
1 octet	1 octet	1 octet	1 octet	2 octets	2 octets

Figure 52 MSAP-MAX_QUEUE_TIME_WRITE.request

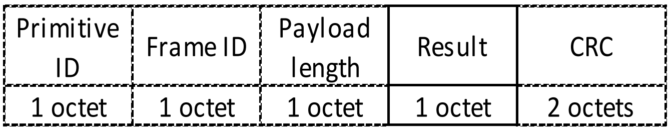
The MSAP-MAX_QUEUE_TIME_WRITE.request frame fields are explained in Table 41.

Table 41 MSAP-MAX_QUEUE_TIME_WRITE.request frame fields

Field Name	**Size**	**Valid Values**	**Description**
Priority	1	0 - 1	Message priority which queuing time to be set. 0 = User traffic class 0, i.e. normal priority 1 = User traffic class 1, i.e. high priority.
Time	4	2 – 65534	Maximum queuing time in seconds. Read instructions in chapter 2.3.15. Default time values after factory reset: Normal priority: 600s = 10 min. High priority: 300s = 5 min.

2.3.15.2. MSAP-MAX_QUEUE_TIME_WRITE.confirm

The MSAP-MAX_QUEUE_TIME_WRITE.confirm is issued in response to the MSAP-MAX_QUEUE_TIME_WRITE.request. The confirmation frame is depicted in Figure 53.



Primitive ID	Frame ID	Payload length	Result	CRC
1 octet	1 octet	1 octet	1 octet	2 octets

Figure 53 MSAP-MAX_QUEUE_TIME_WRITE.confirm

The MSAP-MAX_QUEUE_TIME_WRITE.confirm frame fields are explained in Table 42.

Table 42 MSAP-MAX_QUEUE_TIME_WRITE.confirm frame fields

Field Name	**Size**	**Valid Values**	**Description**
Result	1	0, 3	The return result of the corresponding MSAP-MAX_QUEUE_TIME_WRITE.request: 0 = Success 3 = Failure: Invalid priority or time

2.3.15.3. MSAP-MAX_QUEUE_TIME_READ.request

This request is issued by the application layer to read the current value of the maximum queuing time. The request frame is depicted in Figure 54.

Primitive ID	Frame ID	Payload length	Priority	CRC
1 octet	1 octet	1 octet	1 octet	2 octets

Figure 54 MSAP-MAX_QUEUE_TIME_READ.request

The MSAP-MAX_QUEUE_TIME_READ.request frame fields are explained in Table 43.

Table 43 MSAP-MAX_QUEUE_TIME_READ.request frame fields

Field Name	Size	Valid Values	Description
Priority	1	0 - 1	Message priority which queuing time to be read. 0 = User traffic class 0, i.e. normal priority 1 = User traffic class 1, i.e. high priority.

2.3.15.4. MSAP-MAX_QUEUE_TIME_READ.confirm

The MSAP-MAX_QUEUE_TIME_READ.confirm is issued in response to the MSAP-MAX_QUEUE_TIME_READ.request. The confirmation frame is depicted in Figure 55.

Primitive ID	Frame ID	Payload length	Result	Time	CRC
1 octet	1 octet	1 octet	1 octet	2 octets	2 octets

Figure 55 MSAP-MAX_QUEUE_TIME_READ.confirm

The MSAP-MAX_QUEUE_TIME_READ.confirm frame fields are explained in Table 44.

Table 44 MSAP-MAX_QUEUE_TIME_READ.confirm frame fields

Field Name	Size	Valid Values	Description
Result	1	0, 3	The return result of the corresponding MSAP-MAX_QUEUE_TIME_READ.request: 0 = Success 3 = Failure: Invalid priority
Time	2	2 - 65534	Read value of maximum queuing time in seconds.

2.3.16. MSAP Attributes

The MSAP attributes are specified in Table 45.

Table 45. MSAP attributes

Attribute name	Attribute ID	Type	Size
mStackStatus	1	R	1
mPDUBufferUsage	2	R	1
mPDUBufferCapacity	3	R	1
Reserved	4	-	-
mEnergy	5	R/W	1
mAutostart	6	R/W	1
mRouteCount	7	R	1
mSystemTime	8	R	4

Attribute name	Attribute ID	Type	Size
mAccessCycleRange	9	R/W	4
mAccessCycleLimits	10	R	4
mCurrentAccessCycle	11	R	2
mScratchpadBlockMax	12	R	1
mMulticastGroups	13	R/W	40

2.3.16.1. mStackStatus

Attribute ID	1
Type	Read only
Size	1 octet
Valid values	0x00 – 0x3f or 0x80
Default value	-

The Stack status attribute indicates whether the stack is running or not, and whether it can be started. The stack status is a bit field with individual bits defined in Table 46.

Table 46. Stack status bits

Bit number	Description
0 (LSB)	0: Stack running 1: Stack stopped
1	0: Network address set 1: Network address missing
2	0: Node address set 1: Node address missing
3	0: Network channel set 1: Network channel missing
4	0: Node role set 1: Node role missing
5	0: Application configuration data valid 1: Application configuration data missing (valid only on sink device)
6	Reserved
7 (MSB)	Reserved

2.3.16.2. mPDUBufferUsage

Attribute ID	2
Type	Read only
Size	1 octet
Valid values	0 – *cPDUBufferSize*
Default value	-

The PDUs processed by the stack are stored in a buffer. There is a maximum limit for the number of PDUs that can fit in the buffer (see CSAP *cPDUBufferSize* attribute in section 2.4.4.6). The *mPDUBufferUsage* attribute tells how many PDU items there are in the buffer at the moment.

2.3.16.3. mPDUBufferCapacity

Attribute ID	3
Type	Read only
Size	1 octet
Valid values	0 – *cPDUBufferSize*
Default value	-

The *mPDUBufferCapacity* attribute indicates the number of PDUs that can still fit in the stack PDU buffer at the moment (i.e. *cPDUBufferSize - mPDUBufferUsage*).

2.3.16.4. mEnergy

Attribute ID	5
Type	Read and write
Size	1 octet
Valid values	0 – *255*
Default value	255

The stack can use in its route cost calculations the state of energy remaining in each node. The state can be set with the *mEnergy* attribute. It is a value between 0 and 255, where 0 corresponds to a state where the node is almost out of energy and 255 corresponds to a state where maximum amount of energy is available.

The value is currently used in route cost calculations with a granularity of 32 units. In other words, the energy value must change by at least 32 units for it to affect the cost calculations.

This attribute is intended to be set by the application periodically, to enable the stack routing layer to use energy parameter in cost and route calculation.

The specifics on how the remaining energy is measured is left for the responsibility of the application due to fact that different power sources and measurement circuits may be used depending on the implementation.

2.3.16.5. mAutostart

Attribute ID	6
Type	Read and write
Size	1 octet
Valid values	0 or 1
Default value	1

The stack auto-start function starts the stack automatically after boot (e.g. when returning from power down or when the stack is internally booted). A value of 1 enables auto-start, 0 disables it.

2.3.16.6. mRouteCount

Attribute ID	7
Type	Read only
Size	1 octet
Valid values	0 or 1
Default value	-

Whenever there is a route to sink, the *mRouteCount* attribute is 1, otherwise 0.

2.3.16.7. mSystemTime

Attribute ID	8
Type	Read only
Size	4 octets
Valid values	0 to 4294967295
Default value	-

The stack keeps track of time in 1/128 s increments. Attribute *mSystemTime* can be used to read the amount of time elapsed from the stack startup. *mSystemTime* wraps back to 0 about every 388 days.

2.3.16.8. mAccessCycleRange

Attribute ID	9

Type	Read and write
Size	4 octets
Valid values	See description below
Default value	Not set

Normally the stack chooses a suitable access cycle automatically, between 2, 4 or 8 seconds, depending on the amount of network traffic. Some applications may need to further limit the access cycle durations in use. Attribute *mAccessCycleRange* can be used to do that.

Two 16-bit values are packed in one 32-bit attribute: top 16 bits contain the maximum access cycle duration, bottom 16 bits contain the minimum access cycle duration. Access cycle durations are expressed in milliseconds, so valid values for minimum and maximum are 2000, 4000 and 8000.

If *mAccessCycleRange* is not set, or maximum > minimum, the stack chooses an appropriate access cycle based on the amount of network traffic. If maximum = minimum, the user can force the access cycle to a specific duration. *mAccessCycleRange* is not set by default. Only a factory reset (see section 2.4.3) can restore *mAccessCycleRange* back to the unset state.

Note: When CSMA-CA mode is set as device role (see chapter2.4.4.4), setting *mAccessCycleRange* cannot be done

2.3.16.9. mAccessCycleLimits

Attribute ID	10
Type	Read only
Size	4 octets
Valid values	See description below
Default value	0x1f4007d0

The *mAccessCycleLimits* attribute can be read to determine the valid values for *mAccessCycleRange* (see section 2.3.16.8). Similarly to *mAccessCycleRange*, two 16-bit values are packed in one 32-bit attribute: top 16 bits contain the maximum valid access cycle duration, bottom 16 bits contain the minimum valid access cycle duration.

In current Wirepas Mesh firmware release, the value of *mAccessCycleLimits* is `(8000 << 16) + 2000`, or 0x1f4007d0, i.e. the minimum valid access cycle duration is 2000 ms and the maximum valid access cycle duration is 8000 ms. (4000 ms is also a valid access cycle duration, but that possibility is not encoded in the value of *mAccessCycleLimits*.)

2.3.16.10. mCurrentAccessCycle

Attribute ID	11
Type	Read only
Size	2 octets
Valid values	2000, 4000, 8000
Default value	-

The *mCurrentAccessCycle* attribute reports the currently used access cycle, in milliseconds.

2.3.16.11. mScratchpadBlockMax

Attribute ID	12
Type	Read only
Size	1 octet
Valid values	-
Default value	-

The attribute *mScratchpadBlockMax* contains the value for maximum number of bytes in an MSAP-SCRATCHPAD_BLOCK request (see section 2.3.13.3).

2.3.16.12. mMulticastGroups

Attribute ID	13
Type	Read and Write
Size	10 * 4 octets
Valid values	See description below
Default value	10 x 0xFFFFFFFF

The attribute *mMulticastGroups* tells in which multicast groups the device belongs to. When data packet is sent with multicast group address (see chapter 2.1), only nodes belonging to that group will receive it. Each group is 4 octets long with following value ranges:

- 0: Don’t care value
- 1-0x00FF FFFF: Belonging to multicast groups 0x8000 0001-0x80FF FFFF
- Other values reserved for future use

Note: Albeit the chapter 2.1 defines the multicast addresses to be in range between 0x80000000-0x80FFFFFF, MSB byte (0x80) is not used in this attribute. As well, it is not possible to declare membership to group 0x80000000 (since 3 LSB bytes would be 0).

The value of the attribute is the collection of 10 groups. All values **must** be given. If device belongs to less than 10 groups, use value 0 of don’t care value. Devices don’t have to belong to any groups (which is default value). Don’t care values can reside at the middle of the sequence. Same value can present multiple times in the sequence. Addresses can reside in the sequence in any order.

When data is transmitted to the multicast groups, all the nodes that belong to that group receive the message. Few examples:

Table 47: mMulticastGroups examples

Value of the attribute	Description
0x01000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000	Device belongs to multicast group 0x8000 0001 only
0x01000000 0x02000000 0x03000000 0x04000000 0x05000000 0x06000000 0x07000000 0x08000000 0x09000000 0x0A000000	Device belongs to multicast groups 0x8000 0001 – 0x8000 000A.
0x0A000000 0x09000000 0x08000000 0x07000000 0x06000000 0x05000000 0x04000000 0x03000000 0x02000000 0x01000000	Device belongs to multicast groups 0x8000 0001 – 0x8000 000A. Values can be in any order
0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000	Device does not belong to any group

Value of the attribute	Description
0x00000000 0x00000000 0x00000000 0x00000000 0x05000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000	Device belongs to multicast group 0x8000 0005 only (note: values can be in any order)
0x01000000 0x01000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000	Device belongs to multicast group 0x8000 0001 only. It is ok to set same multicast group multiple times.
0x01000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000	Invalid length
0x01000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000	Invalid length

2.4. Configuration Services (CSAP)

The configuration services are used to control the stack configuration.

2.4.1. CSAP-ATTRIBUTE_WRITE Service

The CSAP-ATTRIBUTE_WRITE service can be used by the application layer to write the configuration attributes to the stack. The CSAP-ATTRIBUTE_WRITE service includes the following primitives:

- CSAP-ATTRIBUTE_WRITE.request
- CSAP-ATTRIBUTE_WRITE.confirm

The CSAP-ATTRIBUTE_WRITE service primitives' frame formats are identical with the MSAP-ATTRIBUTE_WRITE service (See section 2.3.8). Only differences are used primitive IDs and valid attributes that can be read/written. For valid CSAP attributes see section 2.4.4.

Note: The configuration attributes can only be written when the stack is in stopped state.

2.4.2. CSAP-ATTRIBUTE_READ Service

The CSAP-ATTRIBUTE_READ service can be used by the application layer to read the configuration attributes from the stack. The CSAP-ATTRIBUTE_READ service includes the following primitives:

- CSAP-ATTRIBUTE_READ.request
- CSAP-ATTRIBUTE_READ.confirm

The CSAP-ATTRIBUTE_READ service primitives' frame formats are identical with the MSAP-ATTRIBUTE_READ service (See section 2.3.9). Only differences are used primitive IDs and valid attributes that can be read/written. For valid CSAP attributes see section 2.4.4.

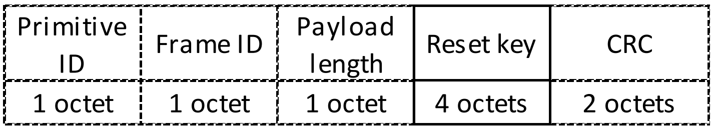
2.4.3. CSAP-FACTORY_RESET Service

The persistent attributes can be cleared using the CSAP-FACTORY_RESET service. The CSAP-FACTORY_RESET service includes the following primitives:

- CSAP-FACTORY_RESET.request
- CSAP-FACTORY_RESET.confirm

2.4.3.1. CSAP-FACTORY_RESET.request

The CSAP-FACTORY_RESET.request issued by the application layer when the persistent attributes should be cleared. The CSAP-FACTORY_RESET.request frame is depicted in Figure 56.



Primitive ID	Frame ID	Payload length	Reset key	CRC
1 octet	1 octet	1 octet	4 octets	2 octets

Figure 56. CSAP-FACTORY_RESET.request frame

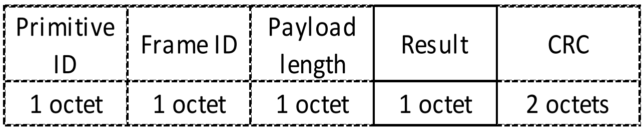
The CSAP-FACTORY_RESET.request frame fields (solid border in the figure) are explained in Table 48.

Table 48. CSAP-FACTORY_RESET.request frame fields

Field Name	Size	Valid Values	Description
ResetKey	4	0x74 0x49 0x6F 0x44 (“DoIt” in ASCII)	Special key value used to verify that user wants to clear persistent values.

2.4.3.2. CSAP-FACTORY_RESET.confirm

The CSAP-FACTORY_RESET.confirm issued by the stack as a response to the CSAP-FACTORY_RESET.request. The CSAP-FACTORY_RESET.confirm frame is depicted in Figure 57.



Primitive ID	Frame ID	Payload length	Result	CRC
1 octet	1 octet	1 octet	1 octet	2 octets

Figure 57. CSAP-FACTORY_RESET.confirm frame

The CSAP-FACTORY_RESET.confirm frame fields are explained in

Table 49.

Table 49. CSAP-FACTORY_RESET.confirm frame fields

Field Name	Size	Valid Values	Description
Result	1	0 – 2	The return result of the corresponding CSAP-FACTORY_RESET.request. The different values are defined as follows: 0 = Success 1 = Failure: Stack in invalid state to clear attributes 2 = Failure: Attempted to use an invalid reset key 3 = Failure: Access denied (see section 2.4.4.21)

2.4.4. CSAP Attributes

The CSAP attributes are specified in Table 50.

Table 50. CSAP attributes

Attribute name	Attribute ID	Type	Size
cNodeAddress	1	R/W	4
cNetworkAddress	2	R/W	3
cNetworkChannel	3	R/W	1
cNodeRole	4	R/W	1
cMTU	5	R	1
cPDUBufferSize	6	R	1
cScratchpadSequence	7	R	1
cMeshAPIVersion	8	R	2
cFirmwareMajor	9	R	2
cFirmwareMinor	10	R	2
cFirmwareMaintenance	11	R	2
cFirmwareDevelopment	12	R	2
cCipherKey	13	W	16
cAuthenticationKey	14	W	16
cChannelLimits	15	R	2
cAppConfigDataSize	16	R	1
cHwMagic	17	R	2
cStackProfile	18	R	2
cOfflineScan	20	R/W	2
cChannelAllocMap	21	R/W	4
cFeatureLockBits	22	R/W	4
cFeatureLockKey	23	W	16

2.4.4.1. cNodeAddress

Attribute ID	1
Type	Read and write
Size	4 octets
Valid values	Any valid *unicast* address
Default value	Not set

Attribute *cNodeAddress* sets the address of the node. See section 2.1 for details.

2.4.4.2. cNetworkAddress

Attribute ID	2
Type	Read and write
Size	3 octets
Valid values	0x000001 – 0xFFFFFE
Default value	Not set

cNetworkAddress, the network address, is used by the radio to detect valid transmissions and to filter out both noise and other transmissions which do not belong to the same network. The network address must be identical for all nodes within the same network. Multiple Wirepas Mesh networks can coexist within an area, if they are configured to use different network addresses.

Due to the way radios detect valid transmissions, some network addresses are better than others. A good network address should have no repetition or patterns. Examples of poor network addresses: 0x000000, 0xFFFFFF, 0xAAAAAA, 0x555555. When choosing a network address for a network, using a 24-bit random number as the network address is recommended.

2.4.4.3. cNetworkChannel

Attribute ID	3
Type	Read and write
Size	1 octet
Valid values	Limits defined by *cChannelLimits* attribute
Default value	Not set

Each network has a default channel, set by the *cNetworkChannel* attribute. The network channel must be identical for all nodes within the same network. Available radio channel range depends on the radio hardware and frequency band of operation. See attribute *cChannelLimits* in section 2.4.4.15.

The network channel is used for finding neighbors in a situation where no neighbors are yet known, e.g. right after the stack has started.

2.4.4.4. cNodeRole

Attribute ID	4
Type	Read and write
Size	1 octet
Valid values	Table 51
Default value	0x82

A Wirepas Mesh network consists of sinks, routing nodes and non-routing nodes. Attribute *cNodeRole* sets the node role. A valid roles are listed in Table 51.

Table 51. Node roles

Value	Base role	Description
0x00	Reserved	
0x01	Sink	A device that is usually connected to a server backbone. All data packets sent to the *AnySink* address end up in here. Similarly, all diagnostic data generated by the network itself is transmitted to a sink device.
0x02	Router Node	A device that is fixed to be capable of routing traffic for other nodes in time slotted mode.
0x03	non-router node	A device that is part of the network but does not route traffic of other nodes in time slotted mode. Mainly used in extremely low-power devices.
0x04 – 0x0F	Reserved	
0x11	CSMA-CA mode Sink	When this is enabled, the sink keeps the receiver enabled all the time when it is not transmitting. Then, the latency on sending data to sink is way faster with the expense on higher power consumption. Intended to be used only with mains-powered devices.
0x12	CSMA-CA mode Router node	When this is enabled, the router node keeps the receiver enabled all the time when it is not transmitting. Then, the latency on sending data to router node is way faster with the expense on higher power consumption. Intended to be used only with mains-powered devices.
0x13	CSMA-CA mode non-router node	When this is enabled, the non-router node keeps the receiver enabled all the time when it is not transmitting. Then, the latency on sending data to router node is way faster with the expense on higher power consumption. Intended to be used only with mains-powered devices.
0x14 – 0x81	Reserved	
0x82	Router node with automatic role selection	A node that is boots up as Router node and capable of routing traffic for other nodes in time slotted mode. Router node is evaluating its role to ensure that there are not too many routing nodes within the radio range. It is highly recommended to enable this in dense and large networks.
0x83	Non-router node with automatic role selection	A node that is boots up as non-router node and without capable of routing traffic for other nodes in time slotted mode. Node is evaluating its role to ensure that there is sufficient amount of routing nodes within the radio range. It is highly recommended to enable this in dense and large networks.
0x83-0x91	Reserved	
0x92	CSMA-CA mode Router node with automatic role selection	A node that is boots up as Router node and capable of routing traffic for other nodes in CSMA-CA mode. Router node is evaluating its role to ensure that there are not too many routing nodes within the radio range. It is highly recommended to enable this in dense and large networks.
0x93	CSMA-CA Non-router node with automatic role selection	A node that is boots up as non-router node and without capable of routing traffic for other nodes in CSMA-CA mode. Node is evaluating its role to ensure that there is sufficient amount of routing nodes within the radio range. It is highly recommended to enable this in dense and large networks.
0x94-0xFF	reserved	

2.4.4.5. cMTU

Attribute ID	5
Type	Read only
Size	1 octet
Valid values	Depends on the radio profile
Default value	-

Attribute *cMTU* contains the Maximum Transmission Unit (MTU) i.e. maximum APDU payload size in octets.

2.4.4.6. cPDUBufferSize

Attribute ID	6
Type	Read only
Size	1 octet
Valid values	-
Default value	-

The PDUs processed by the stack are stored in a buffer. There is a maximum limit for the number of PDUs that can fit in the buffer, as indicated by the *cPDUBufferSize* attribute. See sections 2.3.16.2 and 2.3.16.3 for information about current PDU buffer usage.

2.4.4.7. cScratchpadSequence

Attribute ID	7
Type	Read only
Size	1 octet
Valid values	0 – 255
Default value	-

Attribute *cScratchpadSequence* indicates the sequence number of the OTAP scratchpad present in the node, or 0 if there is no scratchpad stored in the node.

2.4.4.8. cMeshAPIVersion

Attribute ID	8
Type	Read only
Size	2 octets
Valid values	1 – 255
Default value	-

The *cMeshAPIVersion* attribute can be read to determine which version of Wirepas Mesh Dual-MCU API is implemented by the current stack firmware version.

2.4.4.9. cFirmwareMajor

Attribute ID	9
Type	Read only
Size	2 octets
Valid values	-
Default value	-

The *cFirmwareMajor* attribute stores the firmware release major version number, i.e. the first number in the four-part version number: **major**.minor.maintenance.development.

2.4.4.10. cFirmwareMinor

Attribute ID	10
Type	Read only
Size	2 octets
Valid values	-
Default value	-

The *cFirmwareMinor* attribute stores the firmware release minor version number, i.e. the second number in the four-part version number: major.**minor**.maintenance.development.

2.4.4.11. cFirmwareMaintenance

Attribute ID	11
Type	Read only
Size	2 octets
Valid values	-
Default value	-

The *cFirmwareMaintenance* attribute stores the firmware release maintenance version number, i.e. the third number in the four-part version number: major.minor.**maintenance**.development.

2.4.4.12. cFirmwareDevelopment

Attribute ID	12
Type	Read only
Size	2 octets
Valid values	-
Default value	-

The *cFirmwareDevelopment* attribute stores the firmware release development version number, i.e. the fourth number in the four-part version number: major.minor.maintenance.**development**.

2.4.4.13. cCipherKey

Attribute ID	13
Type	Write only
Size	16 octets
Valid values	0x00..00 – 0xFF..FF
Default value	0xFF..FF

Attribute *cCipherKey* sets the key that is used for encrypting radio transmissions. A value of 0xFF..FF means that the key is not set.

It is not possible to read the encryption key back. However, it is possible to detect whether a key is set or not. When reading the key value, the error value 4 (Failure: Invalid attribute value or attribute value not yet set) indicates that key is not set. And error value of 5 (Failure: Write-only attribute) indicates that the key has been set. Writing a key value with all bits set (0xFF..FF) clears the key.

Note: In order for encryption to be enabled, both Cipher and Authentication keys must be set. If only one of them is set, no encryption or authentication is performed.

2.4.4.14. cAuthenticationKey

Attribute ID	14
Type	Write only
Size	16 octets

Valid values	0x00..00 – 0xFF..FF
Default value	0xFF..FF

Attribute *cAuthenticationKey* sets the key that is used for verifying the authenticity of received data. A value of 0xFF..FF means that the key is not set.

It is not possible to read the authentication key back. However, it is possible to detect whether a key is set or not. When reading the key value, the error value 4 (Failure: Invalid attribute value or attribute value not yet set) indicates that key is not set. And error value of 5 (Failure: Write-only attribute) indicates that the key has been set. Writing a key value with all bits set (0xFF..FF) clears the key.

Note: In order for encryption to be enabled, both Cipher and Authentication keys must be set. If only one of them is set, no encryption or authentication is performed.

2.4.4.15. cChannelLimits

Attribute ID	15
Type	Read only
Size	2 octets
Valid values	0x0101 – 0xFFFF
Default value	-

Attribute *cChannelLimits* can be read to determine the allowed range of network channel numbers. See attribute *cNetworkChannel* in section 2.4.4.3.

Lower 8 bits are the first available channel, upper 8 bits are the last available channel. Available radio channel range depends on the radio hardware and frequency band of operation.

2.4.4.16. cAppConfigDataSize

Attribute ID	16
Type	Read only
Size	1 octet
Valid values	80
Default value	-

Size of app config data in octets. See sections 2.3.5-2.3.7.

2.4.4.17. cHwMagic

Attribute ID	17
Type	Read only
Size	2 octets
Valid values	1 – 3
Default value	-

The *cHwMagic* attribute indicates the radio hardware used. Hardware identifiers are listed in Table 52.

Table 52. Hardware identifiers

Value	Radio hardware
0x01	Nordic Semiconductor nRF51x22
0x02	Silicon Labs EFM32 (256 kB Flash / 32 kB RAM)
0x03	Nordic Semiconductor nRF52832 (512 kB Flash / 64 kB RAM)
0x04	Reserved
0x05	Silicon Labs EFR32xG12 (1024 kB Flash / 128 kB RAM)
0x06	Nordic Semiconductor nRF52840 (1024 kB Flash / 256 kB RAM)
0x07	Silicon Labs EFR32xG12 (512 kB Flash / 64 kB RAM)

2.4.4.18. cStackProfile

Attribute ID	18
Type	Read only
Size	2 octets
Valid values	1–3, 5-6
Default value	-

The *cStackProfile* attribute indicates the used frequency band. Frequency bands are listed in Table 53.

Table 53. Frequency bands

Value	Frequency band
0x01	2.4 GHz +4 dBm for Nordic Semiconductor (nRF52832)
0x02	868 MHz +10 dBm for Silicon Labs (EZR32)
0x03	915 MHz +20 dBm USA for Silicon Labs (EZR32)
0x04	Reserved
0x05	917 MHz +14 dBm for Silicon Labs (EZR32)
0x06	Reserved
0x07	Reserved
0x08	865 MHz +20 dBm for Silicon Labs (EZR32)
0x09	2.4 GHz +8 dBm for Silicon Labs (EFR32)
0x10	915 MHz +13 dBm Brazil for Silicon Labs (EZR32)
0x11	915 MHz +16 dBm Australia for Silicon Labs (EFR32)
0x12	2.4 GHz +19 dBm for Silicon Labs (EFR32)
0x13	2.4 GHz +4 dBm for Nordic Semiconductor (nRF52840)
0x14	2.4 GHz +8 dBm for Nordic Semiconductor (nRF52840)
0x15	Reserved

2.4.4.19. cOfflineScan

Attribute ID	20
Type	Read and write
Size	2 octets
Valid values	20 – 600
Default value	CSMA-CA: 30 TDMA: 600

Attribute *cOfflineScan* sets the maximum limit for offline scanning interval value in seconds. When the device does not have a route to the sink, this interval is used to find the route. The scanning interval is a tradeoff between faster rejoin time to the network with the expense of power consumption.

2.4.4.20. cChannelAllocMap

Attribute ID	21

Type	Read and write
Size	4 octets
Valid values	0x00 – 0xFFFFFFFE
Default value	0x11111111

Attribute cChannelAllocMap can be used to dedicate radio channels for devices that have CB-MAC role definition enabled or devices that do not (see section 2.4.4.4).

Each bit in the value represents one radio channel. LSB equals the first available channel (see section 2.4.4.15). If bit is set, the radio channel is dedicated for devices that are configured to CB-MAC mode. If bit is not set, the radio channel is dedicated for devices that are not configured to CB-MAC mode. The default value equals 25% of channels to be dedicated for devices configured to CB-MAC mode.

This attribute is mainly important in dense networks and by using this attribute, the amount of devices within radio range can be maximized. For example: if none of the devices are configured to CB-MAC mode, it is recommended to set this value as 0.

Note: This attribute must be the same throughout the network to operate correctly!

Note: WM FW v3.6.0, v3.6.6, v3.6.7, v3.5.32, v3.5.36 releases and v4.0.xx release onwards cChannelAllocMap attribute does not exists anymore - writing this attribute will return an error code (1 = Failure: Unsupported attribute ID).

2.4.4.21. cFeatureLockBits

Attribute ID	22
Type	Read and write
Size	4 octets
Valid values	See description below
Default value	0xFFFFFFFF

Certain stack features can be disabled by using the *cFeatureLockBits* and *cFeatureLockKey* (see section 2.4.4.22) attributes. Supported feature lock bits are listed in Table 54.

A feature can be disabled by clearing its feature lock bit to zero. By default, no features are disabled, i.e. the feature lock bits are all set. Reserved bits cannot be set to zero. Feature lock is only in effect when a feature lock key is set.

Table 54. Feature lock bits

Lock bits	Feature
0x00000001	Prevent sending data via Dual-MCU API
0x00000002	Reserved
0x00000004	Prevent starting stack via Dual-MCU API
0x00000008	Prevent stopping stack via Dual-MCU API
0x00000010	Prevent setting app config data via Dual-MCU API
0x00000020	Prevent reading app config data via Dual-MCU API
0x00000040	Prevent writing MSAP attributes via Dual-MCU API
0x00000080	Prevent reading MSAP attributes (except *mScratchpadBlockMax* and *mRouteCount*) via Dual-MCU API
0x00000100	Prevent writing CSAP attributes (except *cFeatureLockKey*)
0x00000200	Prevent reading CSAP attributes (except *cScratchpadSequence*) via Dual-MCU API
0x00000400	Reserved
0x00000800	Reserved
0x00001000	Prevent performing factory reset via Dual-MCU API
0x00002000	Prevent scratchpad write operations via Dual-MCU API
0x00004000	Reserved
0x00008000	Prevent reading scratchpad status (including *mScratchpadBlockMax* and *cScratchpadSequence* attributes) via Dual-MCU API
0x00010000	Reserved
0x00020000	Reserved
0x00040000	Reserved
0x00080000	Reserved
0x00100000	Reserved
0x00200000	Prevent reading neighbor information (including *mRouteCount* attribute) via Dual-MCU API
0x00400000	Prevent scanning for neighbors via Dual-MCU API
0x00800000	Reserved
0x01000000	Reserved
0x02000000	Prevent affecting the sink cost via Dual-MCU API
0x04000000	Prevent reading the sink cost via Dual-MCU API
0x08000000	Reserved
0x10000000	Reserved
0x20000000	Prevent sending Remote API requests via Dual-MCU API
0x40000000	Reserved
0x80000000	Prevent participating in OTAP operations

2.4.4.22. cFeatureLockKey

Attribute ID	23
Type	Write only
Size	16 octets
Valid values	0x00..00 – 0xFF..FF
Default value	0xFF..FF

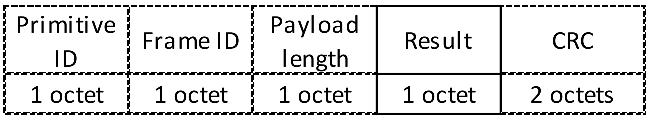
Attribute *cFeatureLockKey* sets the key that is used for enabling the feature lock. A value of 0xFF..FF means that the key is not set. Feature lock bits (see section 2.4.4.21) are only in effect when a key is set.

When a feature lock key is set, it can only be cleared by writing *cFeatureLockKey* with the correct key. This clears the key, i.e. sets a key value with all bits set (0xFF..FF).

It is not possible to read the feature lock key back. However, it is possible to detect whether a key is set or not. When reading the key value, the error value 4 (Failure: Invalid attribute value or attribute value not yet set) indicates that key is not set. And error value of 5 (Failure: Write-only attribute) indicates that the key has been set.

2.5. Response Primitives

All stack indications must be acknowledged by the application using a response-primitive. All the response-primitives have the same frame format as illustrated in Figure 58. Only thing that changes is the Primitive ID. The values of the primitive IDs are listed in Table 3.



Primitive ID	Frame ID	Payload length	Result	CRC
1 octet	1 octet	1 octet	1 octet	2 octets

Figure 58. Generic response-primitive frame

The response-primitive frame fields are explained in Table 55.

Table 55. Generic response-primitive frame fields

Field Name	Size	Valid Values	Description
Result	1	0 or 1	The result field indicates if the application is ready to receive another pending indication (if there are any). The different values are defined as follows: 0 = Do not send more indications 1 = Send next pending indication

2.6. Sequence Numbers

Some Wirepas Mesh stack services, such as the application configuration data service (sections 2.3.5-2.3.7) make use of 8-bit sequence numbers. When new data is entered on the network, the sequence number needs to be incremented, so that nodes can differentiate between old and new data.

Due to the limited numeric range of an 8-bit sequence number, the following wrap-around rule is utilized:

- A is larger than B if (A - B) AND 128 is 0, unless A equals B

For example, values 0 to 127 are considered to be larger than 255, but values 128 to 254 are considered smaller than 255. Likewise, values 1 to 128 are seen as greater than 0, but values 129 to 255 are seen as less than 0.

3. Common Use Cases

This section describes various problem cases with Wirepas Mesh Dual-MCU API as well as guidance on various issues for the user.

3.1. Required Configuration

Each node requires some configuration in order the Wirepas Mesh stack to operate and establish communication. The required services to be configured are explained in Table 56.

Table 56. Required node configuration

Service	See section	Description
CSAP_ATTRIBUTE_WRITE / cNodeAddress	2.4.4.1	A unique device identifier must be set. This is used to distinguish nodes from each other on a network.
CSAP_ATTRIBUTE_WRITE / cNetworkAddress	2.4.4.2	Device network to join. Each device on a network must share the network address.
CSAP_ATTRIBUTE_WRITE / cNetworkChannel	2.4.4.3	One channel in allocated specially for network operations and must be same for each device on a network.
CSAP_ATTRIBUTE_WRITE / cNodeRole (optional)	2.4.4.4	Role of a device must be set. Node can be a sink, a headnode capable of routing or a non-routing subnode.
CSAP_ATTRIBUTE_WRITE / cCipherKey (optional)	2.4.4.13	If encryption of network traffic is desired, a cipher key must be set. Without cipher or authentication keys the encryption is disabled.
CSAP_ATTRIBUTE_WRITE / cAuthenticationKey (optional)	2.4.4.14	If encryption of network traffic is desired, an authentication key must be set. Without cipher or authentication keys the encryption is disabled.
MSAP-APP_CONFIG_DATA_WRITE (optional)	2.3.5	Note: This applies only for sinks! Headnodes or subnodes get their configuration data from the network and cannot have it set directly.
MSAP_STACK_START	2.3.2	Apply settings and begin communicating with other nodes on the network.

4. Annex A: Additional CRC Information

This Annex gives an example CRC implementation and test vectors.

4.1. Example CRC Implementation

```
#include <stdint.h>
// lut table size 512B (256 * 16bit)
static const uint16_t crc_ccitt_lut[] =
{
    0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50a5, 0x60c6, 0x70e7,  \
    0x8108, 0x9129, 0xa14a, 0xb16b, 0xc18c, 0xd1ad, 0xe1ce, 0xf1ef,  \
    0x1231, 0x0210, 0x3273, 0x2252, 0x52b5, 0x4294, 0x72f7, 0x62d6,  \
    0x9339, 0x8318, 0xb37b, 0xa35a, 0xd3bd, 0xc39c, 0xf3ff, 0xe3de,  \
    0x2462, 0x3443, 0x0420, 0x1401, 0x64e6, 0x74c7, 0x44a4, 0x5485,  \
    0xa56a, 0xb54b, 0x8528, 0x9509, 0xe5ee, 0xf5cf, 0xc5ac, 0xd58d,  \
    0x3653, 0x2672, 0x1611, 0x0630, 0x76d7, 0x66f6, 0x5695, 0x46b4,  \
    0xb75b, 0xa77a, 0x9719, 0x8738, 0xf7df, 0xe7fe, 0xd79d, 0xc7bc,  \
    0x48c4, 0x58e5, 0x6886, 0x78a7, 0x0840, 0x1861, 0x2802, 0x3823,  \
    0xc9cc, 0xd9ed, 0xe98e, 0xf9af, 0x8948, 0x9969, 0xa90a, 0xb92b,  \
    0x5af5, 0x4ad4, 0x7ab7, 0x6a96, 0x1a71, 0x0a50, 0x3a33, 0x2a12,  \
    0xdbfd, 0xcbdc, 0xfbbf, 0xeb9e, 0x9b79, 0x8b58, 0xbb3b, 0xab1a,  \
    0x6ca6, 0x7c87, 0x4ce4, 0x5cc5, 0x2c22, 0x3c03, 0x0c60, 0x1c41,  \
    0xedae, 0xfd8f, 0xcdec, 0xddcd, 0xad2a, 0xbd0b, 0x8d68, 0x9d49,  \
    0x7e97, 0x6eb6, 0x5ed5, 0x4ef4, 0x3e13, 0x2e32, 0x1e51, 0x0e70,  \
    0xff9f, 0xefbe, 0xdfdd, 0xcffc, 0xbf1b, 0xaf3a, 0x9f59, 0x8f78,  \
    0x9188, 0x81a9, 0xb1ca, 0xa1eb, 0xd10c, 0xc12d, 0xf14e, 0xe16f,  \
    0x1080, 0x00a1, 0x30c2, 0x20e3, 0x5004, 0x4025, 0x7046, 0x6067,  \
    0x83b9, 0x9398, 0xa3fb, 0xb3da, 0xc33d, 0xd31c, 0xe37f, 0xf35e,  \
    0x02b1, 0x1290, 0x22f3, 0x32d2, 0x4235, 0x5214, 0x6277, 0x7256,  \
    0xb5ea, 0xa5cb, 0x95a8, 0x8589, 0xf56e, 0xe54f, 0xd52c, 0xc50d,  \
    0x34e2, 0x24c3, 0x14a0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,  \
    0xa7db, 0xb7fa, 0x8799, 0x97b8, 0xe75f, 0xf77e, 0xc71d, 0xd73c,  \
    0x26d3, 0x36f2, 0x0691, 0x16b0, 0x6657, 0x7676, 0x4615, 0x5634,  \
    0xd94c, 0xc96d, 0xf90e, 0xe92f, 0x99c8, 0x89e9, 0xb98a, 0xa9ab,  \
    0x5844, 0x4865, 0x7806, 0x6827, 0x18c0, 0x08e1, 0x3882, 0x28a3,  \
    0xcb7d, 0xdb5c, 0xeb3f, 0xfb1e, 0x8bf9, 0x9bd8, 0xabbb, 0xbb9a,  \
    0x4a75, 0x5a54, 0x6a37, 0x7a16, 0x0af1, 0x1ad0, 0x2ab3, 0x3a92,  \
    0xfd2e, 0xed0f, 0xdd6c, 0xcd4d, 0xbdaa, 0xad8b, 0x9de8, 0x8dc9,  \
    0x7c26, 0x6c07, 0x5c64, 0x4c45, 0x3ca2, 0x2c83, 0x1ce0, 0x0cc1,  \
    0xef1f, 0xff3e, 0xcf5d, 0xdf7c, 0xaf9b, 0xbfba, 0x8fd9, 0x9ff8,  \
    0x6e17, 0x7e36, 0x4e55, 0x5e74, 0x2e93, 0x3eb2, 0x0ed1, 0x1ef0  \
};

uint16_t Crc_fromBuffer(uint8_t * buf, uint32_t len)
{
    uint16_t crc = 0xffff;
    uint8_t index;
    for (uint32_t i = 0; i < len; i++)
    {
        index = buf[i] ^ (crc >> 8);
        crc = crc_ccitt_lut[index] ^ (crc << 8);
    }
    return crc;
}
```

4.2. CRC Test Vectors

Table 57. CRC Test Vectors

CRC Input (octets in hex)	CRC Output (octets in hex), LSB first	CRC input length (octets)
None	FF FF	0
0C 01 02 01 00	C2 B1	5
8C 01 05 00 01 00 01 05	48 33	8
0E 02 02 01 00	9D 6E	5
8E 02 08 00 01 00 04 FF FF FF 00	F2 4F	11
0D 03 07 01 00 04 01 00 00 00	8D C4	10
8D 03 01 00	0x0A1F	4

5. References

[1] WP-AN-311 - Non-Router Long Sleep (NRLS)