INTERNATIONAL STANDARD

ISO/IEC 18000-7

Fourth edition
2014-09-01

Information technology — Radio frequency identification for item management —

Part 7: Parameters for active air interface communications at 433 MHz

Technologies de l'information — Identification par radiofréquence (RFID) pour la gestion d'objets —

Partie 7: Paramètres de communications actives d'une interface radio à 433 MHz

Reference number
ISO/IEC 18000-7:2014(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT) see the following URL: Foreword - Supplementary information

The committee responsible for this document is Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 31, *Automatic identification and data capture techniques*.

This fourth edition cancels and replaces the third edition (ISO/IEC 18000-7:2009), which has been technically revised and extended.

ISO/IEC 18000 consists of the following parts, under the general title *Information technology — Radio frequency identification for item management*:

— *Part 1: Reference architecture and definition of parameters to be standardized*

— *Part 2: Parameters for air interface communications below 135 kHz*

— *Part 3: Parameters for air interface communications at 13,56 MHz*

— *Part 4: Parameters for air interface communications at 2,45 GHz*

— *Part 6: Parameters for air interface communications at 860 MHz to 960 MHz General*

— *Part 61: Parameters for air interface communications at 860 MHz to 960 MHz Type A*

— *Part 62: Parameters for air interface communications at 860 MHz to 960 MHz Type B*

— *Part 63: Parameters for air interface communications at 860 MHz to 960 MHz Type C*

— *Part 64: Parameters for air interface communications at 860 MHz to 960 MHz Type D*

— *Part 7: Parameters for active air interface communications at 433 MHz*

Introduction

This part of ISO/IEC 18000 is intended to address radio frequency identification (RFID) devices operating in the 433 MHz frequency band, providing an air interface implementation for wireless, non-contact information system equipment for item management applications. Typical applications operate at ranges greater than one metre.

The RFID system includes a host system and RFID equipment (interrogator and tags). The host system runs an application program, which controls interfaces with the RFID equipment. The RFID equipment is composed of two principal components: tags and interrogators. The tag is intended for attachment to an item, which a user wishes to manage. It is capable of storing a tag serial number and other data regarding the tag or item and of communicating this information to the interrogator. The interrogator is a device, which communicates to tags in its RF communication range. The interrogator controls the protocol, reads information from the tag, directs the tag to store data in some cases, and ensures message delivery and validity. This system uses an active tag.

RFID systems defined by this part of ISO/IEC 18000 provide the following minimum features:

— identify tag in range;

— read data;

— write data or handle read-only systems gracefully;

— selection by group or address;

— graceful handling of multiple tags in the field of view;

— error detection.

This part of ISO/IEC 18000 consists of two modes, Base and Extended. The following simplified differences should be drawn between the two modes:

— Base Mode defined in clause 6 is backwards compatible and includes all features described in the last revision of this part of ISO/IEC 18000 (ISO/IEC 18000-7:2009) with the addition of security features as described in clause 6.7.

— Extended Mode defined in clause 7 is new to this part of ISO/IEC 18000. Extended Mode presents a new communication protocol stack (PHY, MAC and Application layers) and provides an extended feature set that addresses more complex user and deployment requirements.

Substantive differences exist between Base Mode and Extended Mode across all layers of the communication protocol (PHY, MAC and Application). However, both modes may co-exist in any given physical environment.

All parties are directed to consider carefully their use model before determining the most appropriate mode.

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this document may involve the use of patents concerning radio frequency identification technology.

ISO and IEC take no position concerning the evidence, validity and scope of these patent rights.

The holders of these patent rights have assured ISO and IEC that they are willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statements of the holders of these patent rights are registered with ISO and IEC.

Information on the declared patents may be obtained from:

Patent Holder: Legal Name CISC Semiconductor GmbH Contact for license application: Name & Department Markus Pistauer, CEO Address Lakeside B07 Address 9020 Klagenfurt, Austria Tel. +43(463) 508 808 Fax +43(463) 508 808-18 E-mail m.pistauer@cisc.at URL (optional) www.cisc.at
Patent Holder: Legal Name Impinj, Inc. Contact for license application: Name & Department Stacy Jones Address 701 N 34th Street, Suite 300 Address Seattle, WA 98103, USA Tel. +1 206 834 1032 Fax +1 206 517 5262 E-mail stacy.jones@impinj.com URL (optional) www.impinj.com

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those identified above. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

The latest information on IP that may be applicable to this part of ISO/IEC 18000 can be found at www.iso.org/patents.

Information technology — Radio frequency identification for item management —

Part 7: Parameters for active air interface communications at 433 MHz

1 Scope

This part of ISO/IEC 18000 defines the air interface for radio frequency identification (RFID) devices operating as an active RF tag in the 433 MHz band used in item management applications. It provides a common technical specification for RFID devices that can be used by ISO technical committees developing RFID application standards. This part of ISO/IEC 18000 is intended to allow for compatibility and to encourage inter-operability of products for the growing RFID market in the international marketplace. This part of ISO/IEC 18000 defines the forward and return link parameters for technical attributes including, but not limited to, operating frequency, operating channel accuracy, occupied channel bandwidth, maximum power, spurious emissions, modulation, duty cycle, data coding, bit rate, bit rate accuracy, bit transmission order, and, where appropriate, operating channels, frequency hop rate, hop sequence, spreading sequence, and chip rate. This part of ISO/IEC 18000 further defines the communications protocol used in the air interface.

2 Conformance

The rules for evaluation of RFID device conformity to this part of ISO/IEC 18000 are defined in ISO/IEC TR 18047-7.

2.1 RF emissions general population

Device manufacturers claiming conformance to this part of ISO/IEC 18000 shall declare on their own responsibility that RF emissions do not exceed the maximum permitted exposure limits recommended by either IEEE C95.1:2005 or ICNIRP according to IEC 62369-1. If a device manufacturer is unsure which recommendation is to be cited for compliance, the manufacturer shall declare on their own responsibility to ICNIRP limits.

2.2 RF emissions and susceptibility health care setting

Device manufacturers claiming conformance to this part of ISO/IEC 18000 shall declare on their own responsibility that RF emissions and susceptibility comply with IEC 60601-1-2.

2.3 Command structure and extensibility

This part of ISO/IEC 18000 includes a definition of the structure of command codes between an interrogator and a tag and indicates how many positions are available for future extensions.

Command specification clauses provide a full definition of the command and its presentation.

Each command is labelled as being “mandatory” or “optional”.

The clauses of this part of ISO/IEC 18000 make provisions for “custom” and “proprietary” commands.

2.4 Mandatory commands

A mandatory command shall be supported by all tags that claim to be compliant and all interrogators which claim compliance shall support all mandatory commands.

2.5 Optional commands

Optional commands are commands that are specified as such within this part of ISO/IEC 18000. Interrogators shall be technically capable of performing all optional commands that are specified in this part of ISO/IEC 18000 (although they need not be set up to do so). Tags may or may not support optional commands.

If an optional command is used, it shall be implemented in the manner specified in this part of ISO/IEC 18000.

2.6 Custom commands

Custom commands may be permitted by those applying this part of ISO/IEC 18000, but they are not specified in this part of ISO/IEC 18000.

A custom command shall not solely duplicate the functionality of any mandatory or optional command defined in this part of ISO/IEC 18000 by a different method. An interrogator shall use a custom command only in accordance with the specifications of the tag manufacturer.

2.7 Proprietary commands

Proprietary commands may be permitted by those applying this part of ISO/IEC 18000, but they are not specified in this part of ISO/IEC 18000.

A proprietary command shall not solely duplicate the functionality of any mandatory or optional command defined in this part of ISO/IEC 18000 by a different method. All proprietary commands shall be disabled before the tag leaves the tag manufacturer. Proprietary commands are intended for manufacturing purposes and shall not be used in field-deployed RFID systems.

3 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 8859-1, *Information technology — 8-bit single-byte coded graphic character sets — Part 1: Latin alphabet No. 1*

ISO/IEC 15459 (all parts), *Information technology — Unique identifiers*

ISO/IEC 15963, *Information technology — Radio frequency identification for item management — Unique identification for RF tags*

ISO/IEC TR 18047-7, *Information technology — Radio frequency identification device conformance test methods — Part 7: Test methods for active air interface communications at 433 MHz*

ISO/IEC 19762-1, *Information technology — Automatic identification and data capture (AIDC) techniques — Harmonized vocabulary — Part 1: General terms relating to AIDC*

ISO/IEC 19762-3, *Information technology — Automatic identification and data capture (AIDC) techniques — Harmonized vocabulary — Part 3: Radio frequency identification (RFID)*

IEC 62369-1, *Ed. 1.0, Evaluation of human exposure to electromagnetic fields from short range devices (SRDs) in various applications over the frequency range 0 GHz to 300 GHz — Part 1: Fields produced by devices used for electronic article surveillance, radio frequency identification and similar systems*

IEC 60601-1-2, *Medical electrical equipment — Part 1-2: General requirements for basic safety and essential performance — Collateral standard: Electromagnetic compatibility — Requirements and tests*

ICNIRP Guidelines, *Guidelines for limiting exposure to time-varying electric, magnetic, and electromagnetic fields (up to 300 GHz), International Commission on Non-Ionizing Radiation Protection*

IEEE C95.1:2005, *IEEE Standard for Safety Levels with Respect to Human Exposure to Radio Frequency Electromagnetic Fields, 3 kHz to 300 GHz*

IEEE Std 802.15.4, *IEEE Standard for Local and metropolitan area networks Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)*

4 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 19762-1 and ISO/IEC 19762-3 apply.

5 Symbols and abbreviated terms

For the purposes of this document, all symbols and abbreviated terms given in ISO/IEC 19762-1 and ISO/IEC 19762-3 shall apply.

AES	Advanced Encryption Standard
AES-CBC	Advanced Encryption Standard – Cipher Block Chaining
HMAC	Hash-based Message Authentication Code
LR-WPAN	Low-Rate Wireless Personal Area Network
PKI	Public Key Infrastructure
PMK	Pairwise Master Key
PSK	Pre-shared Key
SHA-1	Secure Hash Algorithm – 1
HB2-128	Hummingbird2 128-bit key cipher

6 433,92 MHz active narrowband specification

6.1 Physical layer

The RF communication link between interrogator and tag shall utilize a narrow band UHF frequency with the following nominal characteristics:

Carrier Frequency	433,92 MHz
Modulation Type	FSK
Frequency Deviation	+/- 50 kHz
Symbol LOW	fc +50 kHz
Symbol HIGH	fc -50 kHz

Data Modulation Rate	27,7 kHz
Wake up Signal	Modulation with 31,25 kHz square wave signal followed by modulation with 10 kHz square wave signal

For detailed physical layer specifications, see section 6.6.

The Wake Up Signal shall be transmitted by the interrogator for a minimum of 2,45 seconds to wake up all tags within communication range. The Wake Up Signal shall consist of a 2,35 to 4,8-second 31,25 kHz square wave modulated signal called the "Wake Up Header" immediately followed by a 0,1-second 10 kHz square wave modulated signal called the "Co-Header." Upon detection and by completion of the Wake Up Signal all tags shall enter into the Ready state awaiting a command from the interrogator. See Figure 1. A tag has two states, awake/ready and asleep. During the ready state, the tags will accept the valid commands from interrogators and respond accordingly. When the tag is asleep, it will ignore all commands.

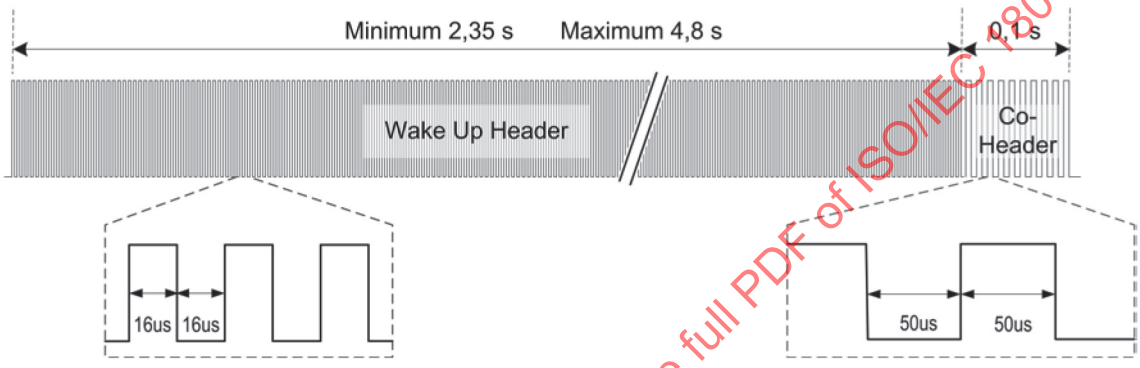


Figure 1 — Wake Up Signal

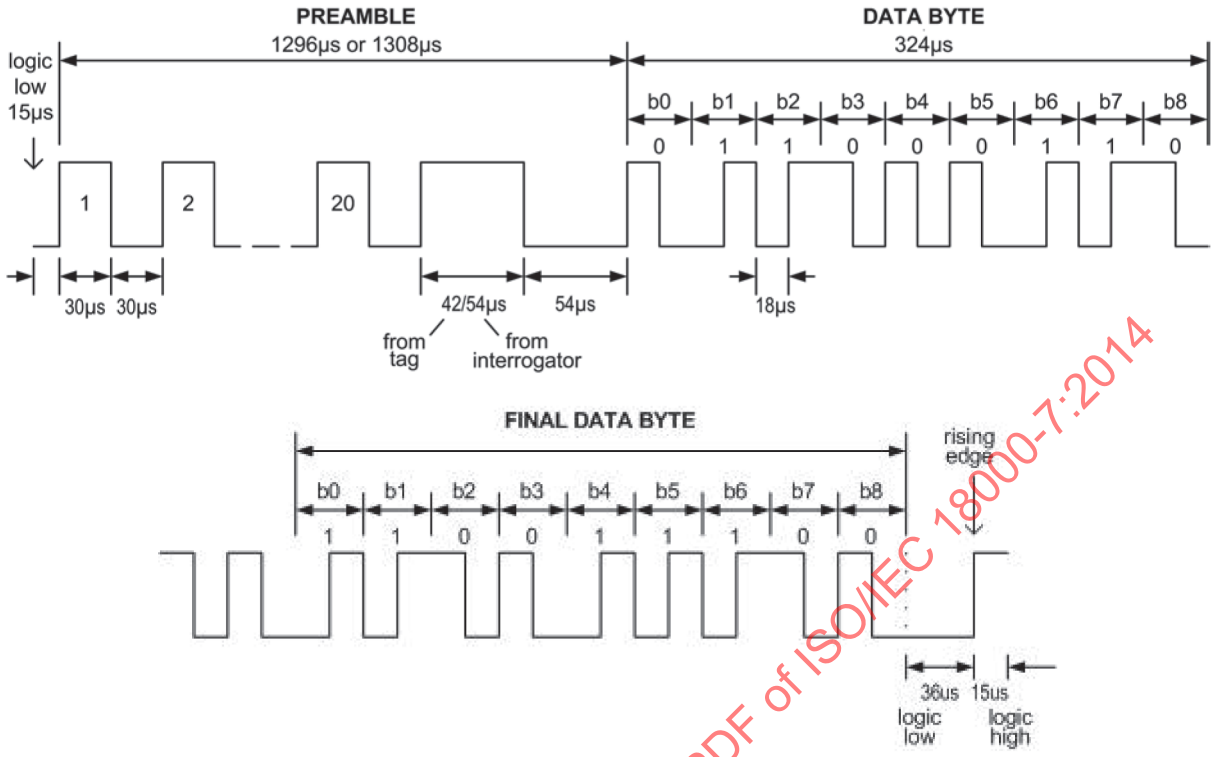
Once awoken, the tag shall stay awake for a minimum of 30 seconds after receipt of the last well-formed message packet consisting of a valid Protocol ID, command code, and CRC values, unless the interrogator otherwise commands the tag to sleep. If no well-formed command message is received within the 30 seconds, the tag will transition to the sleep state and SHALL no longer respond to command messages from Interrogators.

The communication between interrogator and tag shall be of the Master-Slave type, where the interrogator shall initiate communications and then listen for a response from a tag. Multiple response transmissions from tags shall be controlled by the collection algorithm described in 6.4.

6.2 Data Link layer

6.2.1 General

Data between interrogator and tag shall be transmitted in packet format. A packet shall be comprised of a preamble, data bytes and a final end period. The last two level changes of the preamble shall indicate the end of the preamble and beginning of the first data byte. The same two level changes of the preamble also indicate the originator of the data packet. Data bytes shall be sent in Manchester code format. Transmission order shall be most significant byte first; within a byte, the order shall be least significant bit first. Figure 2 illustrates the logic levels for the data communication timing of the preamble and the first byte of a packet.



NOTE Data byte transmitted order is most significant byte first; within each byte the order is least significant bit first. A 15 µs logic low level precedes the first preamble cycle. Byte shown is code 0xC6.

Figure 2 — Data communication timing

6.2.2 Preamble

The preamble shall be comprised of twenty (20) cycles of 60 µs period, 30 µs high and 30 µs low, followed by two final level changes which identifies the communication direction: 42 µs high, 54 µs low (tag to interrogator); or 54 µs high, 54 µs low (interrogator to tag). Refer to Figure 2 above.

6.2.3 Data bytes

Data bytes shall be in Manchester code format, each byte is comprised of 8 data bits and one stop bit. The bit period shall be 36 µs, the total byte period shall be 324 µs. A falling edge in the centre of the bit-time indicates a 0 bit, a rising edge indicates a 1 bit. The stop bit is coded as a zero bit.

6.2.4 Packet end period

A final period of 36 µs of continuous logic low, followed by a logic low to logic high transition, followed by continuous logic high for a minimum of 15 µs shall be transmitted after the last Manchester encoded bit within the packet.

6.2.5 Interrogator-to-tag message format

Tags shall recognize the interrogator-to-tag message format described in Table 1 and Table 2:

Table 1 — Interrogator-to-tag command format (broadcast)

Protocol ID	Packet Options	Packet Length	Session ID	Command Code	Command Arguments	CRC
0x40	1 byte	1 byte	2 bytes	1 byte	N bytes	2 bytes

Table 2 — Interrogator-to-tag command format (point-to-point)

Protocol ID	Packet Options	Packet Length	Tag Manufacturer ID	Tag Serial Number	Session ID	Command Code	Command Arguments	CRC
0x40	1 byte	1 byte	2 bytes	4 bytes	2 Bytes	1 byte	N bytes	2 bytes

See Annex A for other alternative application specific standards, which are identified with their respective Protocol ID.

6.2.5.1 Protocol ID

The protocol ID field allows different application standards based on this part of ISO/IEC 18000 ("derived application standards") to be developed. All derived application standards shall share the same physical layer protocols, but their command/response structure/field and command sets may vary depending on the application. The three basic commands ("Collection with Universal Data Block", "Sleep" and "Sleep All But") defined in this part of ISO/IEC 18000 shall be supported by all derived application standards. All other commands required by this part of ISO/IEC 18000 shall be supported by this part of ISO/IEC 18000 compliant products, but not necessarily by products compliant with derived application standards.

When the interrogator sends out a Wake Up Signal all tags based on the air interface of this part of ISO/IEC 18000 and derived standards shall wake up.

The interrogator may send out various commands as specified by the application. In the event that the interrogator wants to inventory all the active tags within its range, it shall send out a Collection command as defined in this part of ISO/IEC 18000. All tags adhering to this part of ISO/IEC 18000 or derived application standards shall respond to this basic Collection command. A tag shall respond with the collection response defined by the tag's own application data link layer standard (this part of ISO/IEC 18000 or derived standard). The tags shall also accept the Sleep commands ("Sleep" and "Sleep All But") defined in this part of ISO/IEC 18000. The co-existence of this part of ISO/IEC 18000 and derived standards is illustrated in Annex A.

6.2.5.2 Packet Options

Table 3 — Packet options field

Bit							
7	6	5	4	3	2	1	0
Reserved	Reserved	Reserved	Reserved	Reserved	1[a]	0= Broadcast (Tag serial number and Tag manufacturer ID not present) 1= Point to Point (Tag serial number and tag manufacturer ID present)	Reserved

[a] Bit 2 of the "packet oiptions field" has a fixed value of "1" for backwards compatibility.

The Packet Options field, described in Table 3, shall be used to indicate the presence of the Tag serial number and Tag manufacturer ID fields within the command message (packet). As indicated in Table 4, a particular command can be point-to-point or broadcast. The command type is indicated as follows:

— Point-to-point only, Packet Option field Bit 1 must be set to 1.

— Broadcast only and Packet Option field Bit 1 must be set to 0.

Reserved bits are for future use. The default value shall be “0”.

6.2.5.3 Packet Length

The packet length field shall be used to indicate the full length of the message in bytes, from the Protocol ID up to and including the CRC field.

6.2.5.4 Tag Manufacturer ID

The Tag Manufacturer ID is a unique identifier that is issued to each tag manufacturer. The Tag Manufacturer ID is a 16-bit code assigned by the Registration Authority as called out in ISO/IEC 15963. This 16-bit code is a combination of the ISO/IEC 15963 Allocation Class “0001 0001” (most significant byte) and the 8-bit Issuer UID “xxxxxxxx” (least significant byte). For example, if the Issuer UID is assigned as 00000100, the Tag Manufacturer ID would be 00010001 00000100.

The Tag Manufacturer ID format and content shall follow the requirements of unique identifiers as defined in ISO/IEC 15459-1.

The structure and allocation of the Tag Manufacturer ID is described in ISO/IEC 15963 and INCITS 256.

6.2.5.5 Tag Serial Number

The Tag Serial Number is a 32-bit integer that is uniquely assigned to each individual tag during manufacturing. This number cannot be changed and is read only. The Tag Serial Number has no structure and does not contain any information besides uniquely identifying a tag. The Tag Serial Number cannot be reused. Issuance of Tag Serial Numbers may be managed and administered by each manufacturer. The Tag Manufacturer ID and Tag Serial Number together uniquely identify a tag as defined in ISO/IEC 15963. This six-byte combination includes the two-byte Tag Manufacturer ID followed by the Tag Serial Number. An example of the combined data structure for Tag Manufacturer ID and Tag Serial Number is:

00010001 00000100 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx

6.2.5.6 Session ID

The Session ID is a 16-bit integer value that uniquely identifies an interrogator from any other interrogator compliant with this part of ISO/IEC 18000 in the local vicinity. The Session ID of an individual interrogator may be changed without restriction, but its value shall be set to a value not in use by other interrogators compliant with this part of ISO/IEC 18000 in the local vicinity. No two interrogators compliant with this part of ISO/IEC 18000 within RF range of the same tag shall have the same Session ID. At the moment the Session ID is changed in an interrogator, any ongoing communication between that interrogator and any tag shall be terminated. An interrogator that receives a tag message containing a Session ID not equal to its own Session ID shall not transmit any packets over the UHF interface regarding the contents of the tag message. The Session ID 0x0000 is reserved and shall not be used.

6.2.5.7 Command Codes

The Command codes and their function as a Read and/or Write command shall be as listed in Table 4, below. Codes not identified are reserved.

Table 4 — Command codes

Command code + Sub Command Code (R/W)	Command name	Command type	Mandatory/Optional		Description
			Interrogator	Tag	
0x1F / NA	Collection with Universal Data Block	Broadcast	Mandatory	Mandatory	Collects all Tag IDs and Universal Data Block
NA / 0x15	Sleep	Point to Point	Mandatory	Mandatory	Puts tag to sleep
NA / 0x16	Sleep All But	Broadcast	Mandatory	Mandatory	Puts all tags but one to sleep
0x13 / 0x93	User ID	Point to Point	Mandatory	Optional	Sets user assigned ID (1 – 60 bytes)
0x09 / 0x89	Routing Code	Point to point	Mandatory	Mandatory	Reads and writes routing code
0x0C / NA	Firmware Version	Point to Point	Mandatory	Optional	Retrieves manufacturer-defined tag firmware revision number
0x0E / NA	Model Number	Point to Point	Mandatory	Optional	Retrieves manufacturer-defined tag model number
0x60 / 0xE0	Read/Write Memory	Point to Point	Mandatory	Optional	Reads and writes user memory
NA / 0x95	Set Password	Point to Point	Mandatory	Optional	Sets tag password (4 bytes long)
NA / 0x97	Set Password Protect Mode	Point to Point	Mandatory	Optional	Engages/disengages password protection (see section 6.3.4)
NA/ 0x96	Unlock	Point to Point	Mandatory	Optional	Unlocks password protected tag
0x70 / NA	Read Universal Data Block	Point to Point	Mandatory	Mandatory	Reads the Universal Data Block
0x26+0x01	Table Create	Point to Point	Mandatory	Optional	Creates a database table
0x26+0x02	Table Add Records	Point to Point	Mandatory	Optional	Prepares to add new records to the specified database table
0x26+0x03	Table Update Records	Point to Point	Mandatory	Optional	Prepares to modify the specified table records
0x26+0x04	Table Update Fields	Point to Point	Mandatory	Optional	Prepares to update the specified fields of a table record
0x26+0x05	Table Delete Record	Point to Point	Mandatory	Optional	Deletes existing record from the existing database table
0x26+0x06	Table Get Data	Point to Point	Mandatory	Optional	Prepares to retrieve the specified table records
0x26+0x07	Table Get Properties	Point to Point	Mandatory	Optional	Gets total number of records and the maximum number of records the table can hold
0x26+0x08	Table Read Fragment	Point to Point	Mandatory	Optional	Retrieves a block of data from a table as initiated by the Table Get Data command

Table 4 *(continued)*

Command code + Sub Command Code (R/W)	Command name	Command type	Mandatory/Optional		Description
			Interrogator	Tag	
0x26+0x09	Table Write Fragment	Point to Point	Mandatory	Optional	Writes a block of data into a table as initiated by the Table Add Records, Table Update Records, or Table Update fields command
0x26+0x10	Table Query	Broadcast or Point to Point	Mandatory	Optional	Initiates table search based on the specified criteria
0xE1 / NA	Beep ON/OFF	Point to Point	Mandatory	Optional	Turns tag's beeper ON or OFF
0x8E	Delete Writeable Data	Point to Point	Mandatory	Optional	Deletes all allocated writeable data on a tag

The Command Type column indicates whether the command is broadcast (does not include Tag Manufacturer ID and Tag serial number in the message) or point-to-point (includes Tag Manufacturer ID and Tag Serial Number in the message).

For commands requiring a Sub Command Code, the Sub Command Code field is the first byte of the Command Arguments field that follows the Command Code.

6.2.5.8 Command Arguments

Some commands require arguments. For those commands where arguments are defined, argument data shall be supplied with the command. The contents and length of any required arguments are specific to each command. See section 6.3 for details.

6.2.5.9 CRC

A CRC checksum shall be calculated as a 16-bit value for each command message, initialized with all zeroes (0x0000), over all data bytes (excluding preamble) from the protocol ID up to and including any command arguments according to the CCITT polynomial ($x^{16} + x^{12} + x^5 + 1$). The CRC shall be appended to the data included in the command message as a two bytes field. Reference: ITU-T Recommendation V.41 (Extract from the Blue Book), Code-independent error-control system, Appendix I - *Encoding and decoding realization for cyclic code system.*

6.2.6 Tag-to-interrogator message format

The tag-to-interrogator message shall use one of two formats depending on the type of message being transmitted to the Interrogator. The tag shall always respond to a command using one of the response formats described below except in the following situations, for which the tag shall not respond:

— the command is explicitly specified in this part of ISO/IEC 18000 as requiring no response

— the CRC bytes received in the command do not match the CRC bytes that the tag has calculated for the received command packet

— receipt of a broadcast command containing an invalid command code or other error

— the tag is in the asleep state

There are two possible response formats:

— the Broadcast response message format

— the Point-to-Point response message format

6.2.6.1 Broadcast response message format

The message format shown in Table 5 shall be used in response to Interrogator broadcast commands received by tags within the Interrogator's communication range. Broadcast commands are identified in Table 4.

Table 5 — Broadcast response message format

Protocol ID	Tag Status	Packet Length	Session ID	Tag Manufacturer ID	Tag Serial Number	Command Code	Data	CRC
0x40	2 bytes	1 byte	2 bytes	2 bytes	4 bytes	1 byte	N bytes	2 bytes

— **Tag Status:** Indicates various conditions such as response format, tag type, alarm and hardware fault. See section 6.2.6.4, Tag Status, for more details.

— **Packet Length:** Message length in bytes from the Protocol ID field up to and including CRC field.

— **Session ID:** ID of a particular session: An unsigned integer value from 0x0001 to 0XFFFF. The Session ID 0x0000 is reserved and shall not be used.

— **Tag Manufacturer ID:** Unique ID assigned to manufacturer

— **Tag Serial Number:** Unique tag serial number preset during manufacturing

— **Command Code:** Command code (see Table 4) received from the Interrogator

— **Data:** Data returned by the tag as a response to an Interrogator's valid broadcast command request. The value of N, the length of the data in bytes, is specific to the command. In the event that the tag receives an invalid command, no response is sent to the interrogator

— **CRC:** CCITT code check bytes as described in section 6.2.5.9.

6.2.6.2 Point-to-point response message format

This message format, shown in Table 6, shall be returned to the Interrogator as a response to all point-to-point commands, which require the Tag Manufacturer and Serial Number in order to access a particular tag. (Point-to-point commands are identified in Table 4).

Table 6 — Tag–to-interrogator response format (point-to-point)

Protocol ID	Tag Status	Packet Length	Session ID	Tag Manufacturer ID	Tag Serial Number	Command Code	Response Data*	CRC
0x40	2 bytes	1 byte	2 bytes	2 bytes	4 bytes	1 byte	N bytes	2 bytes

*This field is command dependent; some commands may or may not need this field

— **Tag Status**: Indicates various conditions such as response format, tag type, alarm and hardware fault. See section 6.2.6.4, Tag Status, for more details.

— **Packet Length:** Message length in bytes from the Protocol ID field up to and including the CRC field

— **Session ID:** ID of a particular session, an unsigned integer value from 0x0001 to 0xFFFF. The Session ID 0x0000 is reserved and shall not be used.

— **Tag Manufacturer ID**: Unique ID assigned to manufacturer.

— **Tag Serial Number:** Unique tag serial number preset during manufacturing

— **Command Code:** Command code received from the Interrogator

— **Response Data:** Data returned by the tag as a response to an Interrogator's valid command request. The value of N, the length of the data in bytes, is specific to the command. In the event an error is detected, a NACK flag within the Tag Status word will be set and the Response Data will contain an error response as described in subsection 6.2.6.3.

— **CRC:** CCITT code check bytes as described in section 6.2.5.9.

6.2.6.3 Error codes

In response to a point-to-point command a tag may reply with one of the errors listed in Table 7. If multiple errors are detected in a point-to-point command, only the first error is reported. Errors resulting from broadcast commands do not generate responses.

Table 7 — Error code

Error Code	Description
0x01	Invalid Command Code
0x02	Invalid Command Parameter
0x03	Optional Command not Supported
0x04	Not Found
0x06	Can't Create Object
0x08	Authorization Failure
0x09	Object is Read-Only
0x0A	Operation Failed
0x3f	Implementation Dependent
0x40	Stale Token
0x41	Boundary Exceeded

Error response data shall consist of a one-byte error code; possibly a one-byte sub-code, depending on the kind of error; possibly one or more bytes of parameter data, also depending on the error; and an optional, manufacturer-defined number of additional data bytes, as shown in Table 8. In the following error definition sections, the optional, manufacturer-defined data bytes are not shown.

Table 8 — General error format

Error Code	Sub-code	Error Parameter Data	Manufacturer Data
1 byte	1 byte	N bytes	M bytes

— **Error Code:** a value from Table 7 identifying the kind of error

— **Sub-code:** an optional value that further refines the nature of the error and is specific to the kind of error. This field is absent if the error does not define a Sub-code. Sub-codes are specified in the error description subsections below.

— **Error Parameter Data:** N bytes of data, where N is zero or greater, whose existence, length, and content depend on the nature of the error. This field is absent if the error does not define Error Parameter Data. Error specific Error Parameter Data and length N of this field, if any, is specified in the error description subsections below.

— **Manufacturer Data:** M bytes of data, where M is zero or greater, whose existence, field length, and content are at the discretion of the tag manufacturer

6.2.6.3.1 Invalid command code error

Table 9 shows the structure of this error code.

Table 9 — Invalid command code error

Error Code
0x01

This error as defined in Table 9 shall be generated when the tag receives a packet with a Command Code and/or Sub Command Code that is not defined in this part of ISO/IEC 18000.

6.2.6.3.2 Invalid command parameter error

Table 10 shows the structure of this error code.

Table 10 — Invalid command parameter error

Error Code	Sub-code	Parameter Offset
0x02	1 byte	1 byte

— **Sub-code:** a code as shown in Table 11 that describes the error more specifically. Following values are defined:

Table 11 — Invalid command parameter error sub-codes

Sub-code	Sub-error Name	Meaning
0x01	Parameter Out of Range	The value of a parameter is not legal
0x02	Too Few Parameters	There are fewer bytes in the Command Arguments field than expected
0x03	Too Many Parameters	There are more bytes in the Command Arguments field than expected

Parameter offset: the offset in bytes from the beginning of the Command Arguments field where the error was detected.

This error as defined in Table 10 shall be generated when the tag receives a command with invalid or malformed parameters. If more than one parameter is in error, the first invalid parameter shall be reported.

6.2.6.3.3 Optional Command Not Supported

Table 12 shows the structure of this error code.

Table 12 — Optional Command Not Supported error

Error Code
0x03

This error shall be generated when the tag receives an ISO optional command that is not supported on this tag.

6.2.6.3.4 Not found error

Table 13 shows the structure of this error code.

Table 13 — Not found error

Error Code	Sub-code
0x04	1 byte

Sub-code: a code as shown in Table 14 that describes the error more specifically. Following values are defined:

Table 14 — Not found error sub-codes

Sub-code	Sub-error Name	Meaning
0x01	Table Does Not Exist	There is no existing table for the table ID given
0x02	Record Does Not Exist	There are fewer records than the record number given
0x03	Field Does Not Exist	There are fewer fields than the field number given

6.2.6.3.5 Can't create object error

Table 15 shows the structure of this error code.

Table 15 — Can't create object error

Error Code	Sub-code
0x06	1 byte

— **Sub-code:** a code as shown in Table 16 that describes the error more specifically. The following values are defined:

Table 16 — Can't create object error sub-codes

Sub-code	Sub-error Name	Meaning
0x02	Table Already Exists	The requested table ID is already in use
0x03	Out of Memory	There is insufficient memory in the tag to create the requested table
0x04	Table ID Reserved	The table ID provided is reserved, and not available for assignment to a table.

This error as shown in Table 15 shall be generated upon an unsuccessful attempt to create a database table.

6.2.6.3.6 Authorization failure error

Table 17 shows the structure of this error code.

Table 17 — Authorization failure error

Error Code
0x08

This error as shown in Table 17 shall be generated upon an invalid attempt to access a tag feature protected by a password or authorization method.

6.2.6.3.7 Object is read-only error

Table 18 shows the structure of this error code.

Table 18 — Object is read-only error

Error Code
0x09

This error as shown in Table 18 shall be generated upon an attempt to modify some tag data entity for which the tag does not allow modifying operations.

6.2.6.3.8 Operation Failed error

Table 19 shows the structure of this error code.

Table 19 — Operation Failed error

Error Code	Sub-code
0x0A	1 byte

— **Sub-code:** a code as shown in Table 20 that describes the error more specifically. The following values are defined:

Table 20 — Operation Failed error sub-codes

Sub-code	Sub-error Name	Meaning
0x01	Write Failure	The Memory write operation failed.
0x02	Erase Failure	The Memory erase operation failed.
0x03	Memory Consistency	Memory corruption has been detected
0x04	Other Failure	Operation failed for other reason

This error as shown in Table 19 shall be generated upon the failure of a valid command to complete properly. This error shall only be reported if the command failed to complete and no other error has been reported.

6.2.6.3.9 Implementation dependent error

Table 21 shows the structure of this error code.

Table 21 — Implementation dependent error

Error Code	Sub-code
0x3F	1 byte

This error code as shown in Table 21 shall be reserved for tag manufacturers and applications to define for tag behaviour errors not covered by this part of ISO/IEC 18000. At a minimum, the tag implementation shall include a Sub-code field. Sub-code and any additional fields of the error are left to the tag manufacturer and applications to specify.

6.2.6.3.10 Stale Token error

Table 22 shows the structure of this error code.

Table 22 — Stale Token error

Error Code
0x40

This error as shown in Table 22 shall be generated by the tag when a submitted Request Token is invalid due to an intervening modification that was made to the table for which the Request Token applies. These modifications include invocations of the following commands: Table Add Records, Table Update Records, Table Update Fields, and Table Delete Record.

6.2.6.3.11 Boundary exceeded error

Table 23 shows the structure of this error code.

Table 23 — Boundary exceeded error

Error Code	Sub-code
0x41	1 byte

— **Sub-code:** a code as shown in Table 24 that describes the error more specifically. The following values are defined:

Table 24 — Boundary exceeded error sub-codes

Sub-code	Sub-error Name	Meaning
0x01	Table Full	The table has been filled to the create-time allotment
0x02	Record Does Not Exist	The record has not been added yet
0x03	Fragment Overrun	The write operation completed with still more to write
0x04	Field Does Not Exist	The field does not exist

This error as shown in Table 23 and sub-code shown in Table 24 shall be generated upon an attempt to access a record outside of a valid boundary.

6.2.6.4 Tag status

The Tag Status field shown in Table 25, included in all tag–to-interrogator messages, shall consist of the following information:

Table 25 — Tag status field format

Bit							
15	14	13	12	11	10	9	8
Mode field				Alarm	Reserved	Reserved	Acknowledge-ment1 = NACK 0 = ACK
Bit							
7	6	5	4	3	2	1	0
Reserved		Tag type			Reserved	Reserved	Service bit

Note reserved fields are set to a value of “0”.

— **Mode field** indicates the format (response to Broadcast command or response to Point-to-Point command) of the response data from the tag. The list of possible values is shown in Table 26.

Table 26 — Tag status field format

Mode field	Mode format code (bit 15 - 12)
Broadcast Command	0000
Point to Point Command	0010

— **Alarm** is intended as a general status bit indicating a non-command related reportable condition. If set ('1'), an alarm condition has been detected by the tag. The interpretation, actions to retrieve data and clear the alarm bit is defined by the tag vendor.

— **Acknowledgment,** when clear ('0'), indicates that the tag has received a valid command (CRC ok and all fields valid) from the Interrogator and processed the command successfully. If set ('1'), the command was invalid or the tag encountered an error during the processing of the command. Note that as described in section 6.2.6, the tag issues no response in the case of a CRC error.

— **Tag type** is a value assigned by, and meaningful only to, the tag manufacturer. The manufacturer can use this value to indicate manufacturer-defined special features.

— **Service** bit when set ('1') indicates that the tag has detected a hardware-related fault condition. Additional information on the hardware fault condition may be retrieved with the Hardware Fault Status UDB element.

6.3 Tag commands

6.3.1 Collection with Universal Data Block (UDB)

The Collection with Universal Data Block command shall be used to collect Tag Manufacturer ID and Tag Serial Numbers with the contents of a specified UDB data block. The format of the Collection with Universal Data Block (Collection with UDB) command shall be as shown in Table 27.

Table 27 — Collection with Universal Data Block command

Command Code	Windows size	Max Packet Length	UDB Type Code
0x1F	2 bytes	1 byte	1 byte

— **Window Size:** the number of 573 ms intervals to use for listening for tag responses in the collection algorithm. See 6.4 for an explanation of the collection algorithm. Encoded as an unsigned 16-bit integer, with a valid range of 1 to 512.

— **Max Packet Length:** an integer in the range 20 to 255 inclusive that specifies the maximum value that a tag can use as the Packet Length field of it's response. Tags may select a different reply packet length as long as the length does not exceed the value of Max Packet Length. This parameter may be used to tune performance or to limit RF transmission times for compliance with regional RF regulatory requirements. The value 20, the size of a minimum tag response packet (the length 20 includes 15 bytes for response packet overhead, 1 byte for the UDB Type Code value, 2 bytes for the Total UDB Length value and 2 bytes for the Requested Offset value), indicates no bytes of the UDB should be included in the tag response.

— **UDB Type Code:** identifies the requested UDB type. See Table 40 for a list of defined UDB types.

The tag shall select a random time slot based upon the Window Size and Max Packet Length values received (see 6.4). The tag shall respond in the time slot with the Collect with Universal Data Block broadcast response message as shown in Table 28.

When this command is received the tag shall save all requested UDB data and ensure no change to the UDB data until all data has been sent.

Table 28 — Collection with Universal Data Block response

Command Code	UDB Type Code	Total UDB Length	Requested Offset	UDB data
0x1F	1 byte	2 bytes	2 bytes	N bytes

— **UDB Type Code**: identifies the requested UDB type.

— **Total UDB Length**: the total length, in bytes, of UDB data on the tag for the selected UDB Type.

— **Requested Offset**: the tag shall reply with the value zero for its response to a Collection with UDB command. All Collection with UDB commands will begin at the implied offset of zero and the tag shall respond with data beginning at the first byte of the requested UDB block and confirm this offset value with the value 0 for the Requested Offset field.

— **Universal Data Block data**: an initial portion of the Universal Data Block.

6.3.1.1 Universal Data Block

The Universal Data Block contains zero, one or more data elements which are referred to as TLD (Type, Length, Data) Elements, and are formatted as shown in Table 29. Each TLD element is identified with a UDB Element Type ID (see Table 30). Non-present or zero length data elements shall not be included in the Universal Data Block. For example, if the length of the User ID is zero, no part of the User ID TLD shall be included in the UDB.

Table 29 — TLD element format

UDB Element Type ID	Length	Data
1 byte	1 byte	N bytes
UDB Element Type ID (see Table 30)	N (length of Data in bytes)	Data bytes

— **UDB Element Type ID:** identifies Data element, UDB Element Type IDs are defined in Table 30.

— **Length:** number of bytes in length of Data element.

— **Data:** the informational content of the TLD, such as a Routing Code or User ID.

The values for the UDB Element Type ID shall be as shown in Table 30.

Table 30 — UDB Element Type ID values

UDB Element Type ID (1 byte)	Description	Note
0x00 - 0x0A	Reserved	
0x10	Routing Code	The routing code as specified within this document
0x11	User ID	User ID as specified within this document
0x12	Optional Command List	A list of command codes for optional commands supported on this tag
0x13	Memory Size	Total and available memory on this tag
0x14	Table Query Size	The total number of Table Query elements supported on this tag
0x15	Table Query Results	Results for the previously executed Table Query
0x16	Hardware Fault Status	Hardware reset count, Watchdog reset count and Hardware Fault bitmap (including low battery flag) to provide additional information when the “service” bit is set in the tag Status word
0x17 - 0x7F	Reserved	These elements are reserved for future tag data elements

Table 30 *(continued)*

UDB Element Type ID (1 byte)	Description	Note
0x80 - 0xFE	Future extension	Reserved for future use
0xFF	Application Element	Application extensions

The Routing Code UDB Element (0x10) shall be as shown in Table 31.

Table 31 — Routing Code UDB Element

UDB Element Type ID	Length	Data
1 byte	1 byte	N bytes
0x10	N	Routing code data

The User ID UDB Element (0x11) shall be as shown in Table 32.

Table 32 — User ID UDB Element

UDB Element Type ID	Length	Data
1 byte	1 byte	N bytes
0x11	N	User ID data

The Optional Command List Element (0x12) shall be as shown in Table 33. The data returned in this TLD element is a list of one-byte command code values for the optional commands that are implemented on this tag.

Table 33 — Optional Command List Element

UDB Element Type ID	Length	Data
1 byte	1 byte	N bytes
0x12	N	N 1-byte command code values

The Memory Size Element (0x13) shall be as shown in Table 34. The data returned in this TLD is composed of three 4-byte values: the total number of bytes available for Read/Write Memory commands, the total number of bytes allocated for Table database memory, and the number of bytes currently available in the Table database memory (available memory size does not include overhead and simply reports the number of unused memory bytes).

Table 34 — Memory Size Element

UDB Element Type ID	Length	Data		
1 byte	1 byte	12 bytes		
0x13	0x0C	4 bytes	4 bytes	4 bytes
		R/W Memory	Total Table Memory	Available Table Memory

The Table Query Size Element (0x14) shall be as shown in Table 35. The 8-bit unsigned integer value returned in this TLD element represents the number of Table Query elements supported on this tag.

Table 35 — Table Query Size Element

UDB Element Type ID	Length	Data
1 byte	1 byte	1 byte
0x14	0x01	number of Table Query elements supported

The Table Query Results Element (0x15) shall be as shown in Table 36. The data returned in this TLD is available after the successful execution of a Table Query and includes a Query Status value, the Table ID for the queried table, the number of records matched in that table, and the index of the first matching record.

Table 36 — Table Query Results Element

UDB Element Type ID	Length	Data			
1 byte	1 byte	7 bytes			
0x15	0x07	1 byte	2 bytes	2 bytes	2 bytes
		Query Status	Table ID	Number of Records Matched	Index of First Matched Record

The values of the Query Status field shall be as shown in Table 37.

Table 37 — Query Status values

Query Status Value	Description
0x00	The Table Query operation was successful.
0x01	The tag did not execute the query because the tag did not receive a complete sequence of Table Query packets, or a command has been received by the tag that has invalidated any previous query results.
0x02	The tag received a complete sequence of Table Query packets but the tag cannot comply and did not execute the query (e.g. the Table ID is invalid on the tag or a Sequence ID value greater than the maximum number supported by the tag).
0x03	Partial Query Results. The Table Query operation started but has not completed. The Number of Records matched and Index of First Matched Record field represent partial results of the Query.
0x04–0xFF	Reserved.

The Hardware Fault Status Element (0x16) shall be as shown in Table 38. The data returned in this TLD is composed of three 1-byte values: the lifetime count of hardware resets, the lifetime count of Watchdog (firmware) resets, and the Hardware Fault bitmap. The Hardware Fault Bitmap is defined as shown in Table 39.

Table 38 — Hardware Fault Status Element

UDB Element Type ID	Length	Data		
1 byte	1 byte	3 bytes		
0x16	0x03	1 byte	1 byte	1 byte
		lifetime count of hardware resets	lifetime count of firmware resets	Hardware Fault Bitmap

Table 39 — Hardware Fault Bitmap

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
reserved	reserved	reserved	reserved	reserved	reserved	Memory Corruption Detected	Low Battery Detected

Where:

— **Low Battery Detected (bit 0):** when set (‘1’) indicates that the tag battery is “low”. The exact meaning of “low” is implementation defined.

— **Memory Corruption Detected (bit 1):** when set ('1') indicates that the tag has detected a memory hardware fault condition.

— **Reserved (bits 2-7):** reserved for future use.

A UDB Type is a predefined collection of UDB Element Types. The Collection with UDB and Read UDB commands include a UDB Type argument that allows an application to select one of the available predefined collections of UDB data. All UDB Types may include additional Application Extension TLD Elements following the required TLD Elements. The values of the UDB Type shall be as shown in Table 40.

Table 40 — UDB Types

UDB Type	Description	UDB Elements included for this UDB Type
0x00	Transit data	Routing Code UDB element (Element Type 0x10), User ID UDB element (Element Type 0x11) and any Application defined UDB elements.
0x01	Capability data	Optional Command element (Element Type 0x12), Memory Size element (Element Type 0x13) and Table Query Size element (Element Type 0x14) and any Application defined UDB elements.
0x02	Query results	Table Query Results element (Element Type 0x15) and any Application defined UDB elements.
0x03	Hardware Fault data	Hardware Fault Status element (Element 0x16) and any Application defined UDB elements.

The Universal Data Block may optionally include one or more UDB Application Extension Blocks each encapsulating one or more TLDs, which are uniquely identified by the included Application ID (see Table 41). Any individual tag may support the extensions defined by multiple vendors (with appropriate licensing if required).

Table 41 — UDB Application Extension Block format

Application Extension Type ID	Application Extension Length	Application ID TLD Element	Application TLD Elements
1 byte	1 byte	N bytes	M bytes
0xFF	N + M bytes	TLD containing the Application ID Type and Application ID value	one or more Application defined TLDs

Where:

— **Application Extension Type ID:** The Application Extension Type ID defined in Table 30. This Application Extension ID identifies that all TLDs included within this UDB Application Block are identified by the included Application ID.

— **Application Extension Length:** The full length of UDB Application Extension Block in bytes, including the Application ID TLD, and the combined lengths of the included Application TLD elements.

— **Application ID TLD Element:** The Application ID TLD Element must be formatted as described in Table 29 and consists of an Application ID Type, a one-byte length field and a data field containing the Application ID value for the entity responsible for defining the following Application defined TLD elements. Application ID Types are defined as in Table 42.

— **TLD Elements:** A series of one or more TLDs each consisting of a Type ID byte defined by the included Application ID, a one-byte length field and a data field. The TLD Type IDs are defined solely by the Application identified, and are not required to be made public. All of the included TLDs must be formatted as described in Table 29, except that the Type ID is assigned by the manufacturer rather than this part of ISO/IEC 18000. All of the included TLDs must fit completely within the Application Element Length byte count.

Table 42 — Application ID TLD Types

Application ID TLD Type code	Application ID TLD value
0x00	Manufacturer ID - the Application ID is the 16-bit Tag Manufacturer ID assigned by the Registration Authority as called out in ISO/IEC 15963.
0x01	Routing Code - The Application ID is the Tag Data Routing Code as defined in ISO 17363.
0x02 - 0xFF	Reserved

See Figure 3 for an example Universal Data Block of UDB Type 0x00. The example includes a Routing Code element, a User ID element and an Application extension block with two application extension elements.

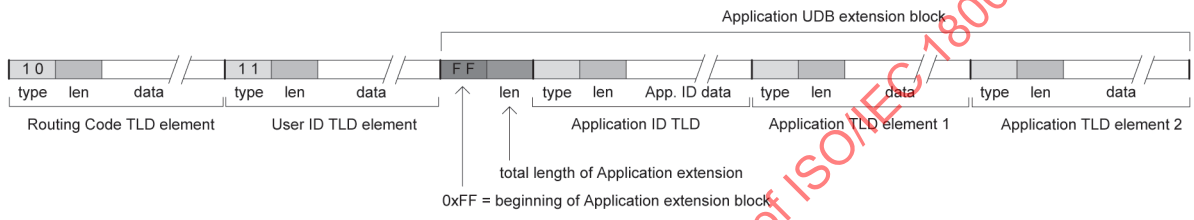


Figure 3 — Example Universal Data Block - UDB Type 0x00

6.3.2 Sleep

To put a tag to Sleep the command in Table 43 shall be sent (written) to the tag.

Table 43 — Sleep command format (Write)

Command code
0x15

Upon receiving the **Sleep** command in Table 43, the tag shall enter the Sleep state. The tag shall not respond to this command and shall ignore any subsequent commands until the tag is woken again by the Wake Up Signal.

6.3.3 Sleep all but

To put all except one tag to Sleep the command in Table 44 shall be sent (written) to the tag.

Table 44 — Sleep all but command format (Write)

Command code	Tag Manufacturer ID	Tag Serial Number
0x16	2 bytes	4 bytes

— **Tag Manufacturer ID**: the Tag Manufacturer ID of the tag which should remain awake following the Sleep All But command.

— **Tag Serial Number:** the Tag Serial Number of the tag which should remain awake following the Sleep All But command.

The Sleep All But command is a broadcast command used to place all tags into the sleep state, as with the Sleep command of section 6.3.2, except for the one tag that matches the provided Tag Manufactures ID and Tag Serial Number. Upon receiving this command, all tags except the one tag that matches the provided Tag Manufactures ID and Tag Serial Number shall enter “sleep” state.

The tags shall not respond to this command.

6.3.4 Security commands

Access to tag write commands shall be guarded by a *password protection* mechanism that application software can command the tag to engage or disengage (see Figure 4). If password protection is engaged, those write commands shall be *non-accessible unless the tag is unlocked*; that is, they will not perform their usual operations but rather respond with an Authorization Failure error. If the password protection is disengaged, the commands are *accessible* – they behave as described in the corresponding sections of this part of ISO/IEC 18000 without the possibility of an Authorization Failure error. Password protection is engaged and disengaged by means of the Set Password Protect Mode command described in section 6.3.4.2. Password protection is disengaged by default.

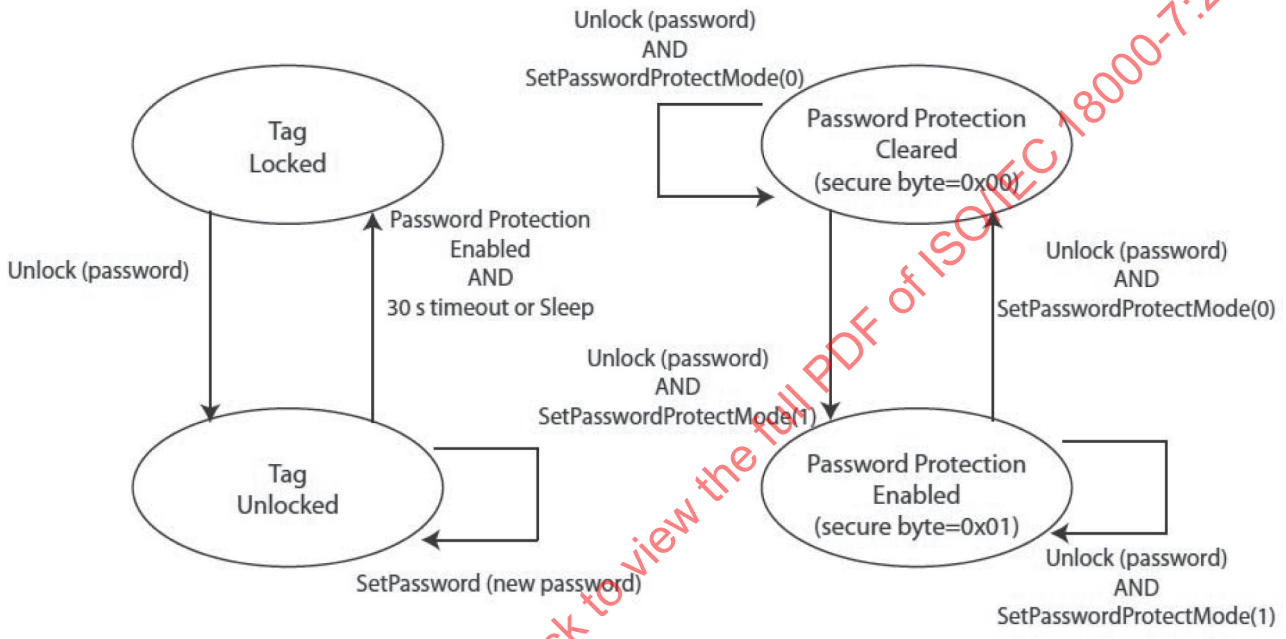


Figure 4 — Tag security state machines

While password protection is engaged, application software can command the tag to enter the *unlocked* state temporarily. While a tag is unlocked, the password-protected write commands shall be accessible. Any time the tag enters the sleep state (either the tag receives a "Sleep" or "Sleep All But" command or 30 seconds passes since the last well-formed command has been received), the tag shall return to the *locked* state, in which the password-protected commands shall be non-accessible. The Unlock command of section 6.3.4.3 puts the tag into the unlocked state. There is no command to put a tag into the locked state explicitly.

Table 45 lists the commands that are affected by password protection.

Table 45 — Write commands affected by password protection

Command code	Command name	Description
0x93	User ID	Sets user assigned ID (1 – 60 bytes)
0x89	Routing Code	Writes routing code
0xE0	Write Memory	Writes user memory
0x95*	Set Password*	Sets tag password (4 bytes long)
0x97*	Set Password Protect Mode*	Engages/disengages password protection
0x26	Table Create	Creates a database table

Table 45 *(continued)*

Command code	Command name	Description
0x26	Table Add Records	Prepares to add new records to the specified database table
0x26	Table Update Records	Prepares to modify the specified table records
0x26	Table Update Fields	Prepares to update the specified fields of a table record
0x26	Table Delete Record	Deletes existing record from the existing database table
0x26	Table Write Fragment	Writes a block of data into a table as initiated by the Table Add Records, Table Update Records, or Table Update fields command
0x8E	Delete Writeable Data	Deletes all allocated writeable data on a tag,

* These commands behave as though password protection were engaged permanently.

6.3.4.1 Security — Set Password

To set the password of a tag, the command in Table 46 shall be sent (written) to the tag.

Table 46 — Set Password command format (write)

Command code	Password
0x95	4 bytes

— **Password:** a four byte binary value, which shall act as the password for subsequent security commands.

To the Set Password command the tag shall respond with a point-to-point response message (and no data, unless an error is encountered) as shown in Table 47.

Table 47 — Set Password command format (write response)

Command code
0x95

This command sets the tag's password. This command requires tag to be first unlocked with the Unlock command of section 6.3.4.3 before the command can be accessed. The initial value of the tag's password is 0xFFFFFFFF.

The possible error responses shall be as shown in the Table 48.

Table 48 — Set Password command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	**Password** parameter is missing or the wrong length
0x08	Authorization Failure	**Unlock** command not invoked prior to invocation of this command

6.3.4.2 Security — Set Password Protect Mode

To set a tag's Password Protect Mode the command in Table 49 shall be sent (written) to the tag.

Table 49 — Set Password Protect Mode command format (write)

Command code	Secure
0x97	1 byte

— **Secure:** a flag that specifies whether password protection shall be engaged or disengaged. The value 0x01 shall cause password protection to be engaged, the value 0x00 shall cause password protection to be disengaged.

To the Set Password Protect Mode command the tag shall respond with a point-to-point response message (and no data, unless an error is encountered) as shown in Table 50.

Table 50 — Set password Protect Mode command format (write response)

Command code
0x97

This command engages or disengages password protection in the tag. To access this command the tag shall first be unlocked with the Unlock command of section 6.3.4.3 regardless of the state of the tag's password protection.

The possible error responses shall be as shown in Table 51.

Table 51 — Set Password Protect Mode command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	Secure parameter is missing or the wrong length
0x08	Authorization Failure	Unlock command not invoked prior to invocation of this command

6.3.4.3 Security — Unlock

To unlock a tag the command in Table 52 shall be sent (written) to the tag.

Table 52 — Unlock command format (write)

Command code	Password
0x96	4 bytes

— **Password**: a four-byte binary value that was previously defined as the password via the Set Password command.

To the Unlock command the tag shall respond (write response) as shown in Table 53.

Table 53 — Unlock command format (write response)

Command code
0x96

This command unlocks the tag. If the supplied password matches tag's password, the tag shall permit the execution of all commands ordinarily non-accessible because of password protection. The tag shall remain in the unlocked state until it receives the Sleep command, Sleep All But command, or 30 seconds has elapsed since the tag received a command.

The possible error responses shall be as shown in Table 54.

Table 54 — Unlock command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	Password parameter is missing or the wrong length
0x08	Authorization Failure	Incorrect password supplied

6.3.5 Transit information commands

6.3.5.1 User ID

The User ID is a user-readable and writeable memory whose meaning and size (up to 60 bytes) is user defined. The User ID format and content shall follow the requirements of unique identifiers as defined in ISO/IEC 15459-3. Moreover, organisations wishing to allocate unique User ID shall do so according to the rules defined by the accredited issuing agency. Issuing Agencies shall apply to the Registration Authority for registration according to 15459-2.

To retrieve a tag's User ID the command in Table 55 shall be sent to the tag.

Table 55 — User ID command format (read)

Command code
0x13

To the User ID read command the tag shall respond with a point-to-point response message with command code and data as shown in Table 56.

Table 56 — User ID command format (read response)

Command code	User ID Length	User ID
0x13	1 byte	N bytes

— **User ID Length:** the length in bytes of the User ID being returned, where N is between 0 and 60 inclusive.

— **User ID:** contents of the User ID on the tag.

To set a tag's User ID the command in Table 57 shall be sent to the tag.

Table 57 — User ID command format (write)

Command code	User ID Length	User ID
0x93	1 byte	N bytes

— **User ID Length:** the length, N, in bytes, of the User ID, where N is between 0 and 60 inclusive.

— **User ID**: the contents of the User ID

To the User ID write command the tag shall respond with a point-to-point response message with command code (and no data, unless an error is encountered) as shown in Table 58.

Table 58 — User ID command format (write response)

Command code
0x93

This command sets and gets the size and contents of the User ID. In addition to this command, the Collection with UDB and Read Universal Data Block commands also retrieve the User ID, except that when the User ID Length parameter is set to zero, the UDB message will not contain the User ID. The default length of the User ID is zero.

The possible error responses shall be as shown in Table 59.

Table 59 — User ID command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	The length of the User ID parameter does not agree with the User ID Length parameter, or the wrong number of parameter bytes was given, or the User ID Length parameter is greater than the maximum, 60
0x08	Authorization Failure	Command an invalid attempt to access a tag feature protected by a password or authorization method
0x0A	Operation Failed	Tag data corrupted, or internal failure on write of User ID on tag

6.3.5.2 Routing Code

To retrieve a tag's Routing Code the command in Table 60 shall be sent to the tag.

Table 60 — Routing Code command format (read)

Command code
0x09

To the Routing Code read command the tag shall respond with a point-to-point response message with command code and data as shown in Table 61.

Table 61 — Routing Code command format (read response)

Command code	Routing Code Length	Routing Code
0x09	1 byte	N bytes

— **Routing Code Length**: the length in bytes of the Routing Code being returned, where N is between 0 bytes and 50 bytes inclusive.

— **Routing Code**: contents of the Routing Code on the tag.

To set a tag's Routing Code the command in Table 62 shall be sent to the tag.

Table 62 — Routing Code command format (write)

Command code	Routing Code Length	Routing Code
0x89	1 byte	N bytes

— **Routing Code Length:** the length, N, in bytes, of the Routing Code, where N is between 0 and 50 inclusive

— **Routing Code:** the data to be written to Routing Code on the tag

To the Routing Code write command the tag shall respond with a point-to-point response message with command code (and no data, unless an error is encountered) as shown in Table 63.

Table 63 — Routing Code command format (write response)

Command code
0x89

The Routing Code is a user-readable and writable memory whose purpose and size (up to 50 bytes) is user defined. The Routing Code should be used as defined in ISO 17363. Note that the Routing Code is part of the tag's response to the Collection with UDB and Read Universal Data Block commands, except

that when the Routing Code Length parameter is set to zero, the UDB message will not contain the Routing Code. The default length of the Routing Code is zero.

The possible error responses shall be as shown in Table 64.

Table 64 — Routing Code command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	**Routing Code Length** parameter is greater than 50 (maximum length permitted), or the length of the **Routing Code** parameter does not agree with the **Routing Code Length** parameter, or the wrong number of parameter bytes was given
0x08	Authorization Failure	Command an invalid attempt to access a tag feature protected by a password or authorization method
0x0A	Operation Failed	Tag data corrupted, or internal failure on write of User ID on tag

6.3.6 Manufacturing Information Commands

The following two commands enable the tag manufacturer to provide manufacturer-defined, immutable information about a tag.

6.3.6.1 Firmware Version

To retrieve a tag's Firmware Version the command in Table 65 shall be sent to the tag.

Table 65 — Firmware Version command format (read)

Command code
0x0C

To the Firmware version command the tag shall respond with a point-to-point response message with command code and data as shown in Table 66.

Table 66 — Firmware Version command format (read response)

Command code	Firmware version
0x0C	4 bytes

— **Firmware Version:** tag firmware version from the tag, a manufacturer defined immutable value.

The Firmware Version indicates the tag firmware version.

6.3.6.2 Model Number

To retrieve a tag's Model Number the command in Table 67 shall be sent to the tag.

Table 67 — Model number command format (read)

Command code
0x0E

To the Model Number command the tag shall respond with a point-to-point response message with command code and data as shown in Table 68.

Table 68 — Model Number command format (read response)

Command code	Model number
0x0E	2 bytes

— **Model number:** tag model number from the tag, a manufacturer defined immutable value.

The Model Number indicates the tag model number.

6.3.7 Memory commands

A tag may provide one or more bytes of user-readable and writable random-access memory in which the user can store and retrieve user-defined data. This memory is independent of all other data storage concepts (such as User ID and tables) defined in this part of ISO/IEC 18000. Associated with every byte of memory is an unsigned integer *address*, through which that memory byte can be accessed. For B bytes of memory the addresses 0 through B-1 access the full range of memory.

6.3.7.1 Write Memory

To write memory the command in Table 69 shall be sent (written) to the tag.

Table 69 — Write Memory command format (write)

Command Code	Number of Bytes	Start Address	Data
0xE0	1 byte	3 bytes	N bytes

— **Number of Bytes:** N, the number of bytes to write, in the range 1 to 237 inclusive. The number of bytes of data in a Write Memory command message must be no greater than 255 – 18 = 237 (18 is the combined length of the command packet header, the number of bytes field, the start address field and the CRC bytes).

— **Start Address:** the memory address of the first memory byte to write, in the range 0 to the manufacturer-defined maximum address.

— **Data:** the memory contents to write.

To the Write Memory command the tag shall respond with a point-to-point response message with command code (and no data, unless an error is encountered) as shown in Table 70.

Table 70 — Write Memory command format (write response)

Command Code
0xE0

The Write Memory command stores the given data into the user random-access memory for later retrieval with the Read Memory command of the next section.

The possible error responses shall be as shown in Table 71.

Table 71 — Write Memory command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	The length of the Data parameter does not agree with the Number of Bytes parameter, or the wrong number of parameter bytes was given, or the Number of Bytes parameter is outside its legal range, or the Start Address plus Number of Bytes extends beyond the maximum address

Table 71 *(continued)*

Error Code	Error Name	Reason
0x08	Authorization Failure	Command an invalid attempt to access a tag feature protected by a password or authorization method
0x0A	Operation Failed	Tag data corrupted, or internal failure on write of memory on tag

6.3.7.2 Read Memory

To read memory the command in Table 72 shall be sent to the tag.

Table 72 — Read Memory command format (read)

Command Code	Number of Bytes to Read	Start Address
0x60	1 byte	3 bytes

— **Number of Bytes to Read:** the number of bytes to read, in the range 1 to 239 inclusive. The number of bytes of data in a Read Memory command message must be no greater than 255 – 16 = 239 (16 is the combined length of the response packet header, the number of bytes field and the CRC bytes).

— **Start Address:** the memory address of the first memory byte to read, in the range 0 to the manufacturer-defined maximum address.

To the Read Memory command, the tag shall respond with a point-to-point response message with command code, parameter, and data as shown in Table 73.

Table 73 — Read Memory command format (read response)

Command Code	Number of Bytes Actually Read	Data
0x60	1 byte	N bytes

— **Number of Bytes Actually Read:** N, the number of bytes of data returned in the response, which always agrees with Number of Bytes to Read.

— **Data:** the memory contents read from tag memory.

The Read Memory command retrieves from the user random-access memory the requested data previously written with the Write Memory command of the previous section.

The possible error responses shall be as shown in Table 74.

Table 74 — Read Memory command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	The wrong number of parameter bytes was given, or the Number of Bytes to Read parameter is outside its valid range, or the Start Address plus Number of Bytes to Read extends beyond the maximum address
0x0A	Operation Failed	Tag data corrupted, or internal failure on read of memory on tag

6.3.8 Delete Writeable Data

To delete all allocated writeable data on a tag, the command in Table 75 shall be sent to the tag. Data that is permanent on the tag and that is marked non-writeable is left untouched.

Table 75 — Delete Writeable Data

Command code
0x8E

To the Delete Writeable Data command the tag shall respond with a point-to-point response message with command code (and no data, unless an error is encountered) as shown in Table 76.

Table 76 — Delete Writeable Data (response)

Command code
0x8E

This command restores all user-writeable memory to factory defaults. In particular, the following operations are performed:

— The length of the User ID is reset to zero.

— The length of the Routing Code is reset to zero.

— All user database tables are deleted. See 6.3.10 for database table definitions.

— The password shall be reset to 0xFFFFFFFF (initial value).

— Password Protect Mode is reset to disabled mode.

— Any existing database table tokens shall be invalidated.

— The Table Query Results table (Table 0x0000) shall be cleared.

The possible error responses shall be as shown in Table 77.

Table 77 — Delete Writeable Data command errors

Error Code	Error Name	Reason
0x08	Authorization Failure	Command an invalid attempt to access a tag feature protected by a password or authorization method
0x0A	Operation Failed	Internal failure on deleting data on tag

6.3.9 Read Universal Data Block

The Read Universal Data Block command is used to read the Universal Data Block (UDB). As described in section 6.3.1.1 the UDB can become large enough to require multiple Read Universal Data Block commands to retrieve the entire UDB. The Offset into UDB field allows an interrogator to retrieve a specific portion of the complete Universal Data Block. To read the Universal Data Block the Read UDB command in Table 78 shall be sent to the tag.

Table 78 — Read UDB

Command Code	UDB Type Code	Offset into UDB	Max Packet Length
0x70	1 byte	2 byte	1 byte

— **UDB Type Code:** identifies the requested UDB type. See Section 6.3.1.1 for further discussion of the UDB Type field.

— **Offset into UDB:** used by the interrogator to identify a starting offset into the specified UDB. In order to retrieve longer Universal Data Blocks, the interrogator will use multiple Read UDB commands and advance the offset value appropriately after each successfully received tag response

— **Max Packet Length:** an integer in the range 21 to 255 inclusive that specifies the maximum value that a tag can use as the Packet Length field in its response. The value 21 includes the 15 bytes of response packet wrapper, one byte of UDB Type Code, two bytes of Total UDB Length value, 2 bytes for the Requested Offset value and at least one byte of UDB data.

To the Read Universal Data Block command, the tag shall respond with a point-to-point response message with command code, parameters, and data as shown in Table 79.

Table 79 — Read UDB Response

Command Code	UDB Type Code	Total UDB Length	Requested Offset	Universal Data Block
0x70	1 byte	2 bytes	2 bytes	N bytes

— **UDB Type Code**: identifies the requested UDB type.

— **Total UDB Length:** the total length, in bytes, of UDB data on the tag for the selected UDB Type.

— **Requested Offset:** the value provided in the Interrogator's command message.

— **Universal Data Block:** a portion of the Universal Data Block. The contents and format of the Universal Data Block are described in section 6.3.1.

To read the entire UDB, an Interrogator will begin with Offset into UDB set to 0 and Max Packet Length set to the largest acceptable packet size. Tags may select a smaller packet size than the length specified by Maximum Packet Length but may not exceed that value. After successfully receiving the initial portion of the UDB, the Interrogator may continue by advancing the Offset into UDB value to the next unread data byte position and sending a second Read UDB command. The interrogator may continue to read the entire UDB but that it does not have to read the entire UDB.

An Interrogator is not required to retrieve the entire UDB. In addition, the Interrogator is not restricted to send Read UDB commands with any ordered sequence of Offset into UDB values to the tag.

The possible error responses shall be as shown in Table 80.

Table 80 — Read UDB command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	The Offset into UDB parameter is greater than the total length of the specified UDB, or Max Packet Length is less than 21, or the wrong number of parameters bytes was given.

6.3.10 Database table commands

The Database Table commands provide basic database functionality, allowing application software to create one or more tables of varying schemas, perform table updates, and query a table. The Database Table commands provide no mechanism for performing table joins. The schema and maximum number of records of a Database Table is fixed at table creation time.

A table schema consists of a list of field (column) widths, in bytes. Fields are numbered (indexed) sequentially, left to right, starting at 0 for the first field. Every field in a table is untyped; that is, all field value comparisons are performed on a byte-for-byte basis, with equality being established between two fields if all bytes in each field match. One field is considered "less than" a second field if for some byte position *p* in the two fields, all bytes in the byte range 0 to *p*-1 are equal in the two fields, and byte *p* of the first field is less than byte *p* of the second field. In other words, a straight multi-byte value comparison is performed with the first byte being the most significant and the last byte being the least significant.

Table records (rows) are indexed starting at 0 for the first record. The record number (the record index) does not maintain a fixed relationship with a record. When a record is deleted, any remaining records in

the database table are re-numbered and may be different than the record order prior to the Table Delete Record command.

Associated with a database table is a Table ID, an immutable 2-byte value that is assigned at table creation time which uniquely identifies a table among all other tables in the tag.

The database tables can be divided into the following types by Table ID, as shown in Table 81.

Table 81 — Table ID space definitions

Table ID range	Table Type
0x0000	Table Query Results (ISO defined)
0x0001 - 0x7FFF	ISO reserved (for future definition)
0x8000 - 0xBFFF	Solution
0xC000 - 0xFFFF	Manufacturer / Vendor

Table IDs in the "ISO Defined" range are reserved for future inclusion in this part of ISO/IEC 18000. Table ID 0x0000 is reserved for the Query Results table (see section 6.3.10.10).

Table IDs in the "Solution" range are reserved for special features, functions and enhancements. In this region, the database tables are read and written with standard database commands, but the tables may have special functions and can have side effects. Table IDs within the Solution range must have published interfaces, and Table ID numbers shall be defined and assigned by the entity that owns the routing code.

Table IDs in the "Manufacturer/Vendor" range are reserved for vendor proprietary extensions, features and enhancements. In this region, the database tables are read and written with standard database commands, but the tables my have special functionality and can have side effects. Table IDs within the Manufacturer/Vendor range are available for use solely at the vendor's discretion, with no requirements to make public the purpose or use of the interfaces within this Table ID space. Data collected from a "Collect with UDB" command contains data from both the Manufacturers Data Block (MDB) and the Universal Data Block (UDB). The MDB data shall be stored in database tables within the range of the Manufacturer/Vendor Table ID space as described in clause 6.3.10.

Read and Write Tokens

Certain table read and write commands produce a data element called a "token". Tokens provide a way for the tag implementation to abstract sequential data access to data sets larger than may be passed in a single message, and do some amount of error detection and recovery. The write commands (Table Add Records, Table Update Records, and Table Update Fields) declare a start location in logical terms (Table ID, record #, field #) and a count. The read command, Table Get Data, declares only a start location. Upon receipt of one of these commands the tag generates a token value and returns it to the interrogator. Subsequently, the token is passed in a Table Read Fragment or Table Write Fragment command from the interrogator back to the same tag, along with any necessary data (subject to context-dependent size restrictions). The tag then performs the read or write, also subject to context-dependent size restrictions, and generates a new token value. The new token is passed back to the interrogator for use in next Table Read Fragment or Table Write Fragment command.

The value of the token is completely at the discretion of the tag implementer, except for the following requirements.

1. While the interrogator is issuing a series of Table Read Fragment or Table Write Fragment commands, by inspecting the token value the tag shall be able to differentiate the next command in the series from the most recently received command in that series. For example, if an interrogator sends the tag a command to read or write a fragment of data, receives no response from the tag, and then sends the same command again with the intention of reading or writing the same fragment, the tag shall identify it as a retry attempt (by means of the token). See Special Database Retry Situations section below.

2. In response to the last command of the series, as determined by the limits imposed by the Table Add Records, Table Update Records, Table Update Fields, or Table Get Data command that preceded the series, the tag shall return a single-byte token whose value is specifically 0x00. That special value informs the interrogator that the tag considers the series to be complete.

3. A tag shall support the existence of multiple, independent "read tokens", and may support the existence of multiple, independent write tokens. A tag shall support a minimum of two independent read tokens.

A "read token" is a token generated by an invocation of the Table Get Data command and used subsequently in invocations of the Table Read Fragment command. A "write token" is a token generated by an invocation of one of the table write commands and used subsequently in invocations of the Table Write Fragment command. The table write commands are Table Add Records, Table Update Records, and Table Update Record Fields. Supporting multiple, independent read tokens means that an invocation of Table Get Data or Table Read Fragment using one token does not affect the operation of those commands using another token, even if the two tokens are associated with the same table. Supporting multiple, independent write tokens means that an invocation of a Table Write command (Table Add Records, Table Update Records, and Table Update Fields) with one token shall not affect the operation of any other Table Write command with another token, provided that the two tokens are associated with different tables. However, invoking a table write command on a table will invalidate all read and write tokens associated with that table.

The high-order 4 bits of the first byte of the token indicates the length of the token, not including the first byte, so zero indicates a token length of 1 byte (see Table Write Fragment). The Token value of exactly 0x00 is reserved, and indicates an end-of-iteration condition. The structure of a Token field is shown below in Table 82:

Table 82 — Token structure

N: Token Length	Token Data
N value in bits 7-4 [Value of N = 0 – 15]	4 low order bits of Token Length byte, then N bytes

Table commands are categorized as being either a *read* command or a *write* command. The read commands include Table Get Data, Table Get Properties, Table Query, and Table Read Fragment, while the table write commands include Table Create, Table Add Records, Table Update Records, Table Update Fields, Table Delete Record, and Table Write Fragment. For all table write commands, the application will have to rewrite the data on the tag for any error occurs during the table write command operation.

Special Database Retry Situations

For the commands Table Create, Table Add Records, Table Delete Records, Table Read Fragment, and Table Write Fragment special error handling is necessary if the interrogator does not receive the response from a successful completion of the command and, therefore, must do a retry of the command. A retry of the command shall be shall an identical copy of the initial invoked command packet, explicitly using the same Session ID, Command Code, Sub Command Code, Sequence ID or Request Token, Table ID (if used), and Data (if used) as the original. The tag shall determine if a command request is a retry of the previously successful database command by comparing it to the previously received command packet. If the tag identifies a request to be a retry of the previous executed and successful database command then the tag SHALL resend the same response from the previous successful command. Refer to command descriptions for Table Add Records, Table Delete Records, Table Read Fragment, and Table Write Fragment for additional details. Note that other database commands also may incur retry requests and retries should be supported.

6.3.10.1 Table Create

When invoking Table Create the command in Table 83 shall be sent to the tag.

Table 83 — Table Create

Command Code	Sub Command Code	Table ID	Maximum Number of Records	Number of Fields	Length of Each Field
0x26	0x01	2 bytes	2 bytes	1 byte	N bytes

Where:

— **Table ID** indicates the identifier to be assigned to the table. Valid ID range is 0x0001 to 0xFFFF. Table ID 0x0000 is reserved for the Query Results Table.

— **Maximum Number of Records** indicates how many records may ultimately exist in the table in total. Valid range is from 0x0001 to 0xFFFF. The remaining amount of unallocated table memory on the tag may additionally limit the valid range.

— **Number of Fields** the number of the fields, N, per record. Its valid range is 1 to 32.

— **Length of Each Field** is a byte array of length N bytes. Each one-byte element of the byte array indicates the size of a field. The first element of the byte array specifies the length of the first field (index 0), the second element specifies the length of the second field (index 1), and so forth. The length of a field shall lie within the range 1 to 255 inclusive.

To the Table Create command the tag shall respond with a point-to-point response message with command code (and no data, unless an error is encountered) as shown in Table 84.

Table 84 — Table Create response

Command Code
0x26

This command creates a database table with a defined maximum number of records, the record format consisting of a specified number of fields each having a specified length. Initially after creation, the table has no records.

The possible error responses shall be as shown in Table 85.

NOTE If the tag identifies a request of this command to be a retry of the previous command that was executed successfully the tag SHALL NOT execute the request and instead, SHALL resend the same response from the previous successful command.

Table 85 — Table Create command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	A parameter is missing, or the Number of Fields parameter is outside its valid range, or the length of the Length of Field array does not match Number of Fields, or one or more of the Length of Field elements is zero, or the wrong number of parameter bytes was given.
0x06	Can't Create Object	The Table ID is already assigned to an existing table and this is not a retry command, or the tag does not have sufficient memory, or the Table ID is 0x0000. The following sub-codes define the kind of error: 0x02 Object already exists 0x03 Out of Memory 0x04 Reserved

Table 85 *(continued)*

Error Code	Error Name	Reason
0x08	Authorization Failure	Command was attempted with password protection engaged and tag in the locked state
0x0A	Operation Failed	Database is corrupted, or unable to create table regardless of valid command parameters or available memory

6.3.10.2 Table Add Records

When invoking Table Add Records the command in Table 86 shall be sent to the tag.

Table 86 — Table Add Records

Command Code	Sub Command Code	Table ID	Sequence ID	Number of Records
0x26	0x02	2 bytes	1 byte	2 bytes

Where:

— **Table ID** indicates the identifier assigned to the table.

— **Sequence ID** is used to identify unique transactions. For each invocation of this command, the interrogator shall supply a different value for Sequence ID. If the interrogator receives no reply from an invocation of the command (due to a communication error, for example) the interrogator shall retry the Table Add Record command using the same Sequence ID as the unsuccessful attempt. The tag shall verify the Sequence ID is different from the value provided with the last successful Table Add Record command, only then is the table record added.

— **Number of Records** indicates the total number of records to add to the table. Valid range is 1 to the Maximum Number of Records set at the time of table creation (Ref. 6.3.10.1) minus the number of records previously added to the table.

To the Table Add Records command the tag shall respond with a point-to-point response message with command code and data as shown in Table 87.

Table 87 — Table Add Records

Command Code	Token
0x26	N bytes

— Token indicates a value used to iteratively write data to the added records. The Token value of exactly 0x00 is reserved, and indicates an end-of-iteration condition. The structure of a Token field is shown above in Table 82.

This command instructs the tag to prepare to add the specified number of records to the Table. The record contents are written to the table with a sequence of Table Write Fragment commands. This command invalidates any existing tokens for this Table ID. This command also invalidates any Table Query results present in Table 0x0000.

NOTE If the tag identifies a request of this command to be a retry of the previous command that was executed successfully the tag SHALL NOT execute the request and instead, SHALL resend the same response from the previous successful command.

The possible error responses shall be as shown in Table 88.

Table 88 — Table Add Records command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	Number of Records is zero, or the wrong number of parameter bytes was given, or the Sequence ID is the same value used with the previous same command.
0x04	Not Found	There is no database table associated with the specified Table ID
0x08	Authorization Failure	Command an invalid attempt to access a tag feature protected by a password or authorization method
0x09	Object is Read-Only	Table, Record, or Field is not writable; such as Table ID 0x0000, the query results table, which is read only
0x41	Boundary Exceeded	The table is too full to accept an additional Number of Records new records

6.3.10.3 Table Update Records

When invoking Table Update Records the command in Table 89 shall be sent to the tag.

Table 89 — Table Update Records

Command Code	Sub Command Code	Table ID	Starting Record Number	Number of Records
0x26	0x03	2 bytes	2 bytes	2 bytes

Where:

— **Table ID** indicates the identifier assigned to the table.

— **Starting Record Number** indicates the first record to begin updating. Valid range is 0 up to (Number of Records in the Table - 1).

— **Number of Records** indicates the total number of records that will be updated. Valid range is 1 up to (Number of Records in the Table - Starting Record Number).

To the Table Update Records command the tag shall respond with a point-to-point response message with command code and data as shown in Table 90.

Table 90 — Table Update Records response

Command Code	Token
0x26	N bytes

— Token indicates a value used to iteratively write data to the updated records. The Token value of exactly 0x00 is reserved, and indicates an end-of-iteration condition. The structure of a Token field is shown in Table 82.

This command instructs the tag to prepare to update the specified table records. The new record contents are written to the table with a sequence of Table Write Fragment commands. This command invalidates any existing tokens for this Table ID. This command also invalidates any Table Query results present in Table 0x0000.

The possible error responses shall be as shown in Table 91.

Table 91 — Table Update Records command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	**Number of Records** is zero, or the wrong number of parameter bytes was given
0x04	Not Found	There is no database table associated with the specified Table ID
0x08	Authorization Failure	Command an invalid attempt to access a tag feature protected by a password or authorization method
0x09	Object is Read-Only	Table, Record, or Field is not writable; such as Table ID 0x0000, the query results table, which is read only
0x41	Boundary Exceeded	Starting **Record Number plus Number of Records** extends beyond the total number of records in the table

6.3.10.4 Table Update Fields

When invoking Table Update Fields the command in Table 92 shall be sent to the tag.

Table 92 — Table Update Fields

Command Code	Sub Command Code	Table ID	Record Number	Starting Field Number	Number of Fields
0x26	0x04	2 bytes	2 bytes	1 byte	1 byte

Where:

— **Table ID** indicates the identifier assigned to the table.

— **Record Number** indicates the record to update. Valid range is 0 up to (Number of Records in the Table - 1).

— **Starting Field Number** indicates the first field to begin updating. Valid range is from 0 up to (Number of Fields in the Table - 1).

— **Number of Fields** indicates the total number of fields in the specified record that will be updated. Valid range is 1 up to (Number of Fields in the Table - Starting Field Number).

This command instructs the tag to prepare to update the specified fields of a table record. The new field contents are written with a sequence of Table Write Fragment commands. This command can only modify fields within a single record, which is provided as the Record Number. This command invalidates any existing tokens for this Table ID. This command also invalidates any Table Query results present in Table 0x0000.

To the **Table Update Fields** command the tag shall respond with a point-to-point response message with command code and data as shown in Table 93.

Table 93 — Table Update Fields

Command Code	Token
0x26	N bytes

— Token indicates a value used to iteratively write data to the updated records. The Token value of exactly 0x00 is reserved, and indicates an end-of-iteration condition. The structure of a Token field is shown in Table 82.

The possible error responses shall be as shown in Table 94.

Table 94 — Table Update Fields command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	Number of Fields is zero, or the wrong number of parameter bytes was given
0x04	Not Found	There is no database table associated with the specified Table ID
0x08	Authorization Failure	Command an invalid attempt to access a tag feature protected by a password or authorization method
0x09	Object is Read-Only	Table, Record, or Field is not writable; such as Table ID 0x0000, the query results table, which is read only
0x41	Boundary Exceeded	Record Number is greater than or equal to the total number of records in the table, or Number of Fields plus Starting Field Number extends beyond the number of fields in the table

6.3.10.5 Table Delete Record

When invoking Table Delete Record the command in Table 95 shall be sent to the tag.

Table 95 — Table Delete Record

Command Code	Sub Command Code	Table ID	Sequence ID	Record Number
0x26	0x05	2 bytes	1 byte	2 bytes

Where:

— **Table ID** indicates the identifier assigned to the table.

— **Sequence ID** is used to identify unique transactions. For each invocation of this command, the interrogator shall supply a different value for Sequence ID. If the interrogator receives no reply from an invocation of the command (due to a communication error, for example) the interrogator shall retry the Table Delete Record command using the same Sequence ID as the unsuccessful attempt. The tag shall verify the Sequence ID is different from the value provided with the last successful Table Delete Record command, only then is the table record deleted.

— **Record Number** indicates the index number of the record to delete.

To the Table Delete Record command the tag shall respond with a point-to-point response message with command code (and no data, unless an error is encountered) as shown in Table 96.

Table 96 — Table Delete Record

Command Code
0x26

This command instructs the tag to delete a single record from the Table, renumbering the record numbers of the remaining records in such a way as to keep the record numbers contiguous starting with 0x0000 (zero). Following execution of Table Delete Record, the order of the remaining records in the table is undefined, and may be different than the record order prior to the Table Delete Record command.

This command invalidates any existing tokens for this Table ID. To read or write data to the database, a new table write command (Table Add Records, Table Update Records, Table Update Fields) or table read command (Table Get Data) shall be issued.

This command also invalidates any Table Query results present in Table 0x0000.

The possible error responses shall be as shown in Table 97.

If the tag identifies a request of this command to be a retry of the previous command that was executed successfully the tag SHALL NOT execute the request and instead, SHALL resend the same response from the previous successful command.

Table 97 — Table Delete Record command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	The wrong number of parameter bytes was given or the Sequence ID is the same value used with the previous same command.
0x04	Not Found	There is no database table associated with the specified Table ID
0x08	Authorization Failure	Command an invalid attempt to access a tag feature protected by a password or authorization method
0x09	Object is Read-Only	Table, Record, or Field is not writable; such as Table ID 0x0000, the query results table, which is read only
0x0A	Operation Failed	Database is corrupted, or unable to complete record removal
0x41	Boundary Exceeded	Record Number is greater than or equal to the total number of records in the table

6.3.10.6 Table Get Data

When invoking **Table Get Data** the command in Table 98 shall be sent to the tag.

Table 98 — Table Get Data

Command Code	Sub Command Code	Table ID	Starting Record Number	Starting Field Number
0x26	0x06	2 bytes	2 bytes	1 byte

Where:

— **Table ID** indicates the identifier assigned to the table.

— **Starting Record Number** indicates the first record to begin reading.

— **Starting Field Number** indicates the first field to begin reading.

To the Table Get Data command the tag shall respond with a point-to-point response message with command code and data as shown in Table 99.

Table 99 — Table Get Data response

Command Code	Token
0x26	N bytes

Where:

— Token indicates a value used to iteratively read record data. The Token value of exactly 0x00 is reserved, and indicates an end-of-iteration condition. The structure of a Token field is shown in Table 82.

The Table Get Data command instructs the tag to prepare to read data from a database table starting with a specified record and field. A sequence of Table Read Fragment commands performs the actual data reading. Unlike the table write commands Table Add Records, Table Update Records, and Table Update Fields, Table Get Data is an open-ended iteration that terminates either at the application software's choosing or when the end of the table is reached.

The Table Get Data tokens, and subsequent tokens returned by Table Read Fragment are invalidated by any of the following commands: Delete Writeable Data, Table Add Records, Table Update Records, Table Update Fields, Table Delete Record, and Table Write Fragment, which operate on the same Table ID.

The possible error responses shall be as shown in Table 100.

Table 100 — Table Get Data command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	The wrong number of parameter bytes was given
0x04	Not Found	There is no database table associated with the specified Table ID, or Table ID is 0x0000 and there is no query result, either because no query was executed or the query result has been made invalid by an intervening table write command on the table that was queried or by starting a new query.
0x41	Boundary Exceeded	Starting Record Number is greater than or equal to the total number of records in the table, or Starting Field Number is greater than or equal to the number of fields in the table

6.3.10.7 Table Get Properties

When invoking Table Get Properties the command in Table 101 shall be sent to the tag.

Table 101 — Table Get Properties

Command Code	Sub Command Code	Table ID
0x26	0x07	2 bytes

Where:

— **Table ID** indicates the identifier assigned to the table.

The Table Get Properties command retrieves information about the specified table. It retrieves the number of used (filled) records in the table and the maximum number of records defined for the table.

To the Table Get Properties command the tag shall respond as shown in Table 102.

Table 102 — Table Get Properties response

Command Code	Total Number of Records	Maximum Number of Records	Reserved
0x26	2 bytes	2 bytes	1 bytes

Where:

— **Total Number of Records** indicates total number of records in the table.

— **Maximum Number of Records** indicates the maximum number of records specified for the table as specified at table creation by the Table Create command.

— **Reserved** is a byte reserved for future use and shall have the value 0x00.

The possible error responses shall be as shown in Table 103.

Table 103 — Table Get Properties command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	The wrong number of parameter bytes was given
0x04	Not Found	There is no database table associated with the specified Table ID

6.3.10.8 Table Read Fragment

When invoking Table Read Fragment the command in Table 104 shall be sent to the tag.

Table 104 — Table Read Fragment

Command Code	Sub Command Code	Request Token	Requested Read Length
0x26	0x08	N bytes	1 byte

Where:

— **Request Token** is the token from the prior Table Get Data or Table Read Fragment command. The Token value of exactly 0x00 is reserved, and indicates an end-of-iteration condition. The structure of a Token field is shown in Table 82.

— **Requested Read Length** is the requested length of data to return. Valid range is from 1 to 46 bytes.

To the Table Read Fragment command the tag shall respond with a point-to-point response message with command code and data as shown in Table 105.

Table 105 — Table Read Fragment response

Command Code	Response Token	Actual Read Length	Data
0x26	N bytes	1 byte	M bytes

Where:

— **Response Token** is the resulting new token from a successful Table Read Fragment command. The Token value of exactly 0x00 is reserved, and indicates an end-of-iteration condition.

— **Actual Read Length** is the number of bytes of data actually read, and may be less than or equal to Requested Read Length.

— **Data** is the actual data read from the tag database table and is the Actual Read Length bytes long.

The Table Read Fragment command reads a block of data bytes from a database table. The database table contents to be read are inherently identified by the Request Token received from the tag via a prior invocation of the Table Get Data command or a previous invocation of this Table Read Fragment command.

The Table Read Fragment command cannot read beyond the last record of a table. If the initial byte to be read by the Table Read Fragment command is within the table, but the Requested Read Length would reach beyond the end of the last record in the table, the command shall be considered valid, and shall return as Actual Read Length not more than the number of bytes remaining to be read in the table.

The possible error responses shall be as shown in Table 106.

NOTE If the tag identifies a request of this command to be a retry of the previous command that was executed successfully the tag SHALL resend the same response from the previous successful command.

Table 106 — Table Read Fragment command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	Request Token is malformed (as defined by the tag implementation), or Requested Read Length is zero, or the wrong number of parameter bytes was given. Request Token is malformed (as defined by the tag implementation), or Requested Read Length is zero, or the wrong number of parameter bytes was given, or the Request Token is 0x00
0x40	Stale Token	Request Token is properly formed and not 0x00 but is invalid due to an intervening modification of the table(s) to which the Request Token applies. These modifications include invocations of the following commands, associated with the Table ID supplied to the commands: Table Add Records, Table Update Records, Table Update Fields, and Table Delete Record.
0x0A	Operation Failed	Read operation failed or database is corrupted

6.3.10.9 Table Write Fragment

When invoking Table Write Fragment the command in Table 107 shall be sent to the tag.

Table 107 — Table Write Fragment

Command Code	Sub Command Code	Request Token	Data Length	Data
0x26	0x09	N bytes	1 byte	N bytes

Where:

— **Request Token** is the token from the prior Table Add Records, Table Update Records, Table Update Fields, or Table Write Fragment command. The structure of a Token field is shown in Table 82.

— **Data Length** is the length of data to write. Valid range is from 1 to 46 bytes.

— **Data** is the data bytes to be written to the tag database table.

To the Table Write Fragment command the tag shall respond with a point-to-point response message with command code and data as shown in Table 108.

Table 108 — Table Write Fragment response

Command Code	Response Token
0x26	N bytes

Where:

— **Response Token** is the resulting new token from a successful Table Write Fragment command. The Token value of exactly 0x00 is reserved, and indicates an end-of-iteration condition. The structure of a Token field is shown in Table 82.

The Table Write Fragment command writes a block of data bytes to a database table. The database table contents to write are inherently identified by the Request Token received from the tag via a prior invocation of the Table Add Records, Table Update Records, or Table Update Fields command or a previous invocation of this Table Write Fragment command.

This command invalidates any existing tokens for this Table ID. This command also invalidates any Table Query results present in Table ID 0x0000.

The possible error responses shall be as shown in Table 109.

NOTE If the tag identifies a request of this command to be a retry of the previous command that was executed successfully the tag SHALL resend the same response from the previous successful command.

Table 109 — Table Write Fragment command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	Request Token is malformed (as defined by the tag implementation), or Data Length is zero, or the length of the Data parameter does not agree with Data Length, or the wrong number of parameter bytes was given, or the Request Token is 0x00
0x08	Authorization Failure	Command was attempted with password protection engaged and tag in the locked state
0x0A	Operation Failed	Write operation failed or database is corrupted
0x40	Stale Token	Request Token is properly formed and not 0x00 but is invalid due to an intervening modification of the table(s) to which the Request Token applies. These modifications include invocations of the following commands, associated with the Table ID supplied to the commands: Table Add Records, Table Update Records, Table Update Fields, and Table Delete Record.
0x41	Boundary Exceeded	The Data Length for this request would exceed the length declared in the original Table Add Records, Table Update Records, or Table Update Record Fields command

6.3.10.10 Table Query

When invoking Table Query the command in Table 110 shall be sent to the tag. The Table Query command can be sent as either a Broadcast message to all tags simultaneously, or as a Point-to-Point message to a single tag.

Table 110 — Table Query

Command Code	Sub Opcode	Table ID	Sequence ID	Query Element				
				Logical Operator	Logical Operand			
					Field Number	Relational Operator	Comparison Data Length	Comparison Data
0x26	0x10	2 bytes	1 byte	1 byte	1 byte	1 byte	1 byte	N bytes

Where:

— **Table ID** indicates the identifier assigned to the table.

— **Sequence ID** identifies a query element among a sequence of query elements. For a sequence of N query elements, the Sequence ID is N-1 for the first query element, N-2 for the second query element, and so forth, down to 0 (zero) for the Nth query element. The tag shall support a minimum of 4 query elements per sequence; Sequence IDs from 3 down to 0. The actual number of query elements supported on a tag can be retrieved through the UDB Element Type 0x15 (Table Query Size). See Section 6.3.1.1.

— **Logical Operator** defines the role of the current query element within the complete query. The possible values of the logical operator are the ISO/IEC 8859-1 characters ‘C’ (CLEAR), ‘A’ (AND), or ‘O’ (OR).

— **Field Number** indicates index number of the field to match. The Field Number shall be less than the number of fields in the table.

— **Relational Operator** defines the method by which the field contents are compared with Data. The possible values of the relation operator are the ISO/IEC 8859-1 characters '=' (EQUAL), '<' (LESS THAN), '>' (GREATER THAN), or '!' (NOT EQUAL).

— **Comparison Data Length** indicates length of the Comparison Data in bytes. The range of Comparison Data Length is 1 to 32.

— **Comparison Data** specifies the byte array to which the field contents are compared. Comparison Data is **Comparison Data Length** bytes long, and may include the special ISO/IEC 8859-1 prefix '*', the wildcard character.

6.3.10.10.1 Overview of Query Syntax

This command defines a *query element*, one table search criterion among a sequence of such criteria. A complete query conceptually has the form:

{<query element$_1$>} {<query element$_2$>} … {<query element$_N$>}

where each <query element>, of which there is at least one, has the form:

<logical operator> <logical operand>

where <logical operand> has the form:

<field number> <relational operator> <comparison data>

where Logical Operator, Field Number, Relational Operator, and Comparison Data are fields in the Table Query command format shown in Table 110. Logical Operator is one of the ISO/IEC 8859-1 characters 'C', 'A', or 'O', Field Number is a table field, Relational Operator is one of the ISO/IEC 8859-1 characters '=', '<', '>', '!', and Comparison Data is a 1 to 32 byte string of data bytes. The angle brackets (<, >) and curly braces ({, }) in the above syntax serve only as delimiters for the purposes of this discussion and have no syntactic meaning or literal presence in an actual command. A complete query, therefore, is specified as a sequence of Table Query commands.

6.3.10.10.2 Query Elements

Query elements within a complete query are related to one another by their logical operators, which specify how those query elements are aggregated into a compound Boolean expression. A logical operator can be a logical AND, a logical OR, or the special case CLEAR. Logicals AND and OR are left-associative binary operators of equal precedence which have their conventional Boolean meanings, while CLEAR merely indicates that the query element is the first element of the complete query. If CLEAR is the logical operator for any query element, any prior set of query elements are discarded and the current query element is to be regarded as the first query element of a new query. Upon receipt of a valid Query containing a CLEAR, any pre-existing results from any previous query shall be removed; all existing records in Table 0x00 shall be deleted.

The relational operands consist of the database table field identified by the Field Number and the Comparison Data.

6.3.10.10.3 Interpretation of Queries

A complete query is to be interpreted as an expression whose constituents are the logical operator and logical operand of each query element, read left to right. For example, suppose a complete query is composed of four query elements and the logical operands of the first, second, third, and fourth query elements are A, B, C, and D, respectively. The complete query

(CLEAR A) (AND B) (OR C) (AND D)

is to be interpreted as the Boolean expression

CLEAR (((A AND B) OR C) AND D)

where for each record of the table being searched each logical operand evaluates to “true” or “false” values as described below, the Boolean operators combine those values into a single “true” or “false” value in the conventional manner for Boolean operators, and the CLEAR operator has no impact on the Boolean value of the entire expression. If the entire expression evaluates to “true” the table record is included in the query results table, also described below.

6.3.10.10.4 Logical Operands

A logical operand specifies how each record of the table to be searched is to be checked for inclusion in the set of matching records. The field number of the logical operand specifies which field of each record is to be inspected. The comparison data specifies the value to which the field contents are to be compared. And the relational operator specifies the manner in which the field contents and comparison data, the two *relational operands* of the relational operator, are to be compared. Additionally, the first byte of the comparison data affects the nature of the comparison. If that byte is the ISO/IEC 8859-1 character ‘*’, the comparison is a *wildcard comparison*; otherwise, the comparison is a *full-match comparison*.

6.3.10.10.5 Full-Match Comparisons

If the relational operator is ‘=’ for a full-match comparison, the relational operands are compared on a byte-for-byte basis for an exact match. If the bytes at some position in both relational operands do not match, the logical operand evaluates to “false”. If one relational operand is longer than the other, the logical operand evaluates to “false”. Otherwise, the logical operand evaluates to “true”. For example, the following comparison evaluates to “true”.

“abc” ‘=’ “abc”

The following comparisons evaluate to “false”.

“abc” ‘<’ “abc”

“abdb” ‘<’ “abce”

“abc” ‘>’ “abc”

“abc” ‘!’ “abc”

If the **Relation Operator** is ‘!’ for a full-match comparison, the comparison is handled in the same manner as the ‘=’ relation operator but generates the opposite result. Any comparison in which the ‘=’ operator would result in the “true” condition, the ‘!’ operator results in “false”, and any comparison in which the ‘=’ operator would result in the “false” condition, the ‘!’ operator results in “true”. The following comparisons evaluate to “true”.

“abc” ‘!’ “abcd”

“abc” ‘!’ “ABC”

“abc” ‘!’ “abd”

“abc” ‘!’ “ab”

The following comparison results in “false”.

“abc” ‘!’ “abc”

If the Relational Operator is ‘<’ or ‘>’ for a full-match comparison, the relational operands are compared on a byte-for-byte basis as for the ‘=’ operator until the first non-matching byte is found. If no non-matching byte is found, the logical operand evaluates to “false”. The non-matching bytes are compared according to the relational operator. If the operator is ‘<’ and the byte from the field contents is less than the byte from the comparison data, the logical operand evaluates to “true”. If the operator is ‘>’ and the byte from the field contents is greater than the byte from the comparison data, the logical operand evaluates to “true”. For the inequality operators ‘<’ and ’>’, if the length of one relational operand is less

than that of the other relational operand, then the shorter operand is considered for the purposes of comparison to contain an additional final byte whose value is less than the minimum possible value for a byte.

Note that because the comparison data is limited to 32 bytes, for fields greater than 32 bytes in length, full-match comparisons with the '=' operator always evaluate to "false", while full-match comparisons with the '!' operator always evaluate to "true".

For example, the following comparisons evaluate to "true".

"abb" '<' "abc"

"aad" '<' "abc"

"ab" '<' "abc"

"abc" '<' "ad"

"abc" '>' "abb"

"abc" '>' "aad"

"abc" '>' "ab"

"ad" '>' "abc"

"abc" '!' "abd"

"abc" '!' "ab"

The following comparisons evaluate to "false".

"abc" '<' "abc"

"abdb" '<' "abce"

"abc" '>' "abc"

"abc" '!' "abc"

6.3.10.10.6 Wildcard Comparisons

For wildcard comparisons with the relational operator '=', the field contents starting with the first byte and the comparison data starting with the byte after '*' are compared on a sliding basis for a match as in a full-match comparison until the end of the field contents is reached. That is, starting from the beginning of the field contents and sliding to the right a byte at a time until the end of the field contents, Comparison Data Length bytes of field data are compared against the Comparison Data Length bytes of Comparison Data bytes looking for a complete match. If a complete match is found, the comparison is discontinued. If a complete match was found and the Relational Operator was '=', the comparison evaluates to a "true", otherwise it evaluates to a "false". The results for the '!' operator are "false" if the complete match was found, otherwise it evaluates to a "true". Wildcard comparisons with the relational operators '<' and '>' are illegal, as are wildcard comparisons for which the comparison data is the single character '*'.

In the following examples, the first item is the field number, and the last item is the Comparison Data.

The following comparisons evaluate to "true."

"abcde" '=' "*bcd"

"abcbcde" '=' "*bcd"

"abcecd" '!' "*bcd"

The following comparisons evaluate to “false.”

“abcde” ‘!’ “*bcd”

“abce” ‘=’ “*bcd”

6.3.10.10.7 Query Failures

The possible error responses shall be as shown in Table 111.

Table 111 — Table Query command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	Sequence ID is greater than the maximum number of query operators that the tag supports; or Sequence ID is not the same as, or one less than, that of the previous Table Query command AND Logical Operator is not CLEAR; or Table ID is not the same as that of the previous Table Query command AND Logical Operator is not CLEAR; or Comparison Data Length, Logical Operator or Relational Operator are outside their valid range of values; or Data Length is zero; or the length of Data does not agree with Data Length, or the wrong number of parameter bytes was given
0x04	Not Found	There is no database table associated with the specified Table ID
0x41	Boundary Exceeded	Field Number greater than or equal to the number of fields in the table

6.3.10.10.8 Execution of Complete Query

Upon receipt of the final **Table Query** command (which has a Sequence ID of zero), the tag should now have the complete Query criteria. The tag shall execute the complete query on each record in the table identified by Table ID, beginning with Record Number 0 and incrementing through all the records in the table.

The Query Results Table (Table ID 0x0000) contains the complete results of the query operation. The Query Results Table has records with a single two-byte field. Each 2-byte field/record in the Query Results table contains the record number of a matching record in the queried table. The matching record numbers, if any, in the Query Results table, shall increase monotonically. Records in Table 0x0000 shall be returned in response to Table Get Data and Table Read Fragment. The record number of each matching record shall be returned as individual records in MSB first order.

6.3.10.10.9 Point-to-Point and Broadcast Queries

The Table Query command exists as both a point-to-point command and a broadcast command. The value of the Packet Options field, as described in section 6.2.6.1, determines whether the command is broadcast or point-to-point. The tag does not respond with any message to any broadcast Table Query command, even in case of error.

For non-final point-to-point Table Query commands, which have a non-zero Sequence ID, the tag shall verify it received a valid non-final Table Query command. If the command is a valid initial or intermediate Table Query command, the tag shall respond with a point-to-point response message with command code (and no data, unless an error is encountered) as shown in Table 112. This response merely indicates that the tag successfully received a valid query element, so no database query operation results are available or expected.

Table 112 — Intermediate Table Query Response

Command Code
0x26

Upon completion of the point-to-point query command sequence, the tag shall respond with a point-to-point response message with command code and data as shown in Table 113. If the query resulted in no records matched, the number of records matching the criteria shall be zero and the index of the first matched record shall be zero in response to the final point-to-point **Table Query** command.

Table 113 — Final Point-to-Point Intermediate Table Query response

Command Code	Number of Records matched	Index of first matched record
0x26	2 bytes	2 bytes

— **Number of Records Matched** contains the number of records found in the queried table, which meet the complete query criteria. This field shall be formatted as an unsigned 16-bit integer. If no matching records were found, this field shall contain 0.

— **Index of First Matched Record** contains the record number of the first matching record of the queried table, which meets the complete query criteria. If no records were found, this field shall contain 0.

An Interrogator may follow up a sequence of point-to-point Table Query commands by using the Table Get Data and Table Read Fragment commands to retrieve the results from the Query Results Table (Table 0x0000).

6.3.10.10.10 Broadcast Collection with UDB (Query Results UDB)

The broadcast Collection with UDB command may be used to retrieve the query results from tags after the complete sequence of Table Query commands has been transmitted. To retrieve the query results, an Interrogator may send the Collection with UDB command with the UDB Type field set to 0x02 (see Table 36). Tags will return their Tag serial number with the Table Query Results element (see Table 36). The Table Query Results element includes the index of the queried table, the number of matching records and the index of the first matching record. If the query resulted in no records matched, the number of matching records shall be zero and the index of the first matched record shall be zero. The Interrogator may then follow up successful query matches by retrieving the records in the Query Results Table (Table 0x0000) with Table Get Data and Table Read Fragment commands.

6.3.10.10.11 Deleting Table Query Results

The results of the Table Query command are written to the query results table (reserved Table ID 0x0000). Any database command that modifies any of the database tables on the tag (Table Add Records, Table Update Records, Table Update Fields, Table Delete Record) shall force deletion of all records in the query results table. Any subsequent Table Get Properties commands for the query results table shall return 0 as the the number of records currently in the table.

6.3.11 Beep ON/OFF

When invoking Beep ON/OFF the command in Table 114 shall be sent to the tag.

Table 114 — Beep ON/OFF

Command Code	Beeper On/Off
0xE1	1 byte

Where:

— **Beeper On/Off** parameter when 0x01 will turn tag's beeper ON or when set to 0x00 will turn tag's beeper OFF.

To the Beep ON/OFF command the tag shall respond with a point-to-point response message with command code (and no data, unless an error is encountered) as shown in Table 115.

Table 115 — Beep ON/OFF response

Command Code
0xE1

The Beep ON/OFF command turns the tag's beeper on or off. When the tag's beeper is turned on, the beeper stays on until explicitly turned off or until the tag returns to the Sleep state.

The possible error responses shall be as shown in Table 116.

Table 116 — Beep ON/OFF command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	Beeper On/Off parameter is missing or the wrong length or outside its valid range of values

6.3.12 Sensor implementation

The retrieval and transmission of data and control information related to tag-based sensors can be implemented within this part of ISO/IEC 18000. Sensor status and data can be added to a returned Universal Data Block using an Application extension block. Sensor data logs and control information can be read and written using the existing database table commands.

6.3.12.1 UDB application extensions for sensors

Manufacturers can record sensor status in the UDB using the UDB Application Extension Block format (see Table 41 and the related descriptive text in clause 6.3.1.). Manufacturers can record sensor status in the UDB using the UDB Application Extension Block format (see Table 41 and the related descriptive text in clause 6.3.1.).

Depending on the application and the complexity of the sensor implementation, the UDB application extension block provides flexibility on how to report sensor status. Specifically, one or many TLD elements can be defined in the UDB according to the TLD element format described in Table 29.

6.3.12.2 Sensor data storage

Sensor data and control information can be stored in database tables as described in section 6.3.10. Sensor related data can be stored in either the Solution Table ID space or the Manufacturer / vendor Table ID space depending on the application as defined in section 6.3.10.

6.3.12.3 Commands to retrieve sensor status information and sensor data

The following standard commands can be triggered by the interrogator to retrieve sensor status information in the UDB:

— Collection with UDB (command code 0x1F)

— Read UDB (command code 0x70)

Sensor activity logs can be retrieved using the following commands

— Table Get Data (command code 0x26+0x06 followed by a Table Read Fragment (command code 0x26+0x08)

Or

— Table Query (command code 0x26+0x10)

Standard response message formats are sent back to the interrogator with the requested information.

6.3.12.4 Sensor data characteristics and formats

The sensor data reported by the RF tag should follow the formats described in ISO/IEC/IEEE 21451.

NOTE Sensor data formats will also be described in a future International Standard (ISO/IEC 24753).

6.3.12.5 Physical interface between the sensor and RF tag

When the tag and sensors are discrete/separate units, the physical interface between the sensor and the RF tag should be as described in ISO/IEC/IEEE 21451-7

6.4 Tag collection and collision arbitration

This standard specifies the method by which the interrogator shall identify and communicate with one or more tags present in the operating field of the interrogator over a common radio frequency channel. Communications specified include methods to: identify a tag, read data from a tag, write data to a tag and command the tag to perform a specific function. Tags do not transmit unless commanded to do so by the interrogator, and an interrogator can communicate with tags individually, or with the tag population as a whole.

In the following discussion, the terms *all tags* and *tag population* refer only to tags within the operating field of the interrogator.

General explanation

The tag collection process is used to identify tags in the operating field of the interrogator. This is an iterative process that includes methods for coordinating responses from the tag population and handling collisions, which occur when multiple tags transmit at the same time. The entire tag collection process is referred to as a Complete Collection Sequence.

Tag Collection

Figure 5 shows a complete collection sequence consisting of a wakeup period (WP) followed by a series of collection periods (CP). Each collection period consists of a synchronization period (SP), a listen period (LP), and an acknowledge period (AP).

— Wakeup Period (WP) – the time period in which the interrogator transmits one or more Wake Up Signals to bring all tags to the ready state. This Wake Up Signal is defined in section 6.1. The Wakeup Period is transmitted only once during a collection sequence.

— Collection Period (CP) – the time periods in which the tags are actually identified and acknowledged. A sequence of Collection Periods is used, repeating until all responding tags have been identified. Each Collection Period consists of a Synchronization Period, a Listen Period, and an Acknowledge Period.

— Synchronization Period (SP) – the time period in which the interrogator sends a broadcast *collection* command to the tag population in the operating field of the interrogator. Each tag shall synchronize its timing with the end of the packet reception Interrogator broadcast command.

— Listen Period (LP) – the time period in which the interrogator waits for responses from tags. The listen period is divided into Time Slots (TS) which are time windows for tags to respond. Each tag selects a random Time Slot for its response, and delays its response to fit into the chosen Time Slot.

— Acknowledge Period (AP) – the time period in which the interrogator acknowledges responding tags and may optionally retrieve additional data from a tag. For each tag identified by the interrogator during the previous Listen Period, the interrogator optionally collects additional data from the tag, and then commands the tag to sleep using the Sleep command.

Collection Period

As shown in Figure 6, the collection period consists of a synchronization period, listen period, and acknowledge period. The listen period is further divided into multiple Time Slots as shown in Figure 5.

The interrogator's collection command provides the duration of the Listen Period as the Window Size parameter. Listen Period is computed:

Listen Period Duration = (Windows Size * 57.3 milliseconds), rounded up to the nearest millisecond

The Listen Period is divided into Time Slots, which are individual time windows for tag response messages. The Time Slot duration is the time necessary to transmit the maximum length tag response message plus a slot guard time to cover timing inconsistencies between tags and the interrogator. The maximum tag message size is provided as the *Max Packet Length* parameter, which is included in a *Collection with Universal Data Block* command. The Time Slot duration is computed:

Time Slot Duration = (324 µs/byte * Max Packet Length) + 3332 µs, rounded up to the nearest millisecond

324 µs/byte is the transmit time for each byte (8 data bits plus one stop bit at 36 µs per bit).

3332 µs is the duration of the tag preamble (1296 µs) plus packet end period of 36 µs, plus a 2 millisecond slot guard time.

The Number of Time Slots is computed by the number of Time Slots, which can fit into the Listen Period. It is computed:

Number of Time Slots = (Listen Period Duration / Time Slot Duration), rounded down to nearest integer

Wake Up Period

The following is a step-by-step description of the Complete Collection Sequence. See the timing diagram in Figure 5.

Wake Up Period:

— Interrogator: transmits the Wake Up Signal to all tags in the operating field of the interrogator.

— Tag: wake up into the ready state and listen for commands from the interrogator.

Collection Period:

— Interrogator: transmit a Collection with Universal Data Block command to the tags. This command includes the Window Size and Max Packet Length parameters.

— Tag: If in the ready state, receives and decodes the Collection with Universal Data Block command, computes Listen Time Duration, Time Slot Duration and Number of Time Slots. Randomly selects a Time Slot for its response to interrogator and delays until the start of the selected time slot.

— Interrogator: computes Listen Time Duration, Time Slot Duration and Number of Time Slots. Begins listening for tag responses.

— Tag: at beginning of selected time slot, transmits the response message to the interrogator. The tag shall issue only one response during a Collection Period.

— Interrogator: receives valid tag response with a correct CRC, or a collision (if more than one tag chose the same response time slot), or the interrogator receives no tag response because no tag responded during that time slot (empty time slot).

— Interrogator: continues listening for the remainder of time slots, until the end of the Listen Period.

— Interrogator: for each tag from which the interrogator received a valid response message, the interrogator may send point-to-point *Read Universal Data Block* commands to retrieve any remaining UDB from the tag, and then the interrogator shall send a point-to-point Sleep command to the tag.

— Tag: will respond to any point-to-point Read Universal Data Block commands, which may be received from the interrogator.

— Tag: on receiving a Sleep Command, shall leave the ready state and shall not respond to further commands from the interrogator until after the tag receives a Wake Up Signal.

— Interrogator: If during the preceding Collection Period, the Interrogator senses that there are too many collisions or that there is a significant amount of empty airtime, the Interrogator shall adapt the communication channel bandwidth by transmitting control parameters in the Collection command. If there are too many collisions, the Interrogator shall use a larger Window Size parameter in the *Collection with Universal Data Block* command. If there are few or no collisions, then the Interrogator shall reduce the Window Size parameter.

— Interrogator: if no tags were identified and no collisions were detected, the interrogator shall repeat the collection period for a minimum of one and up to a maximum of three consecutive empty collection periods.

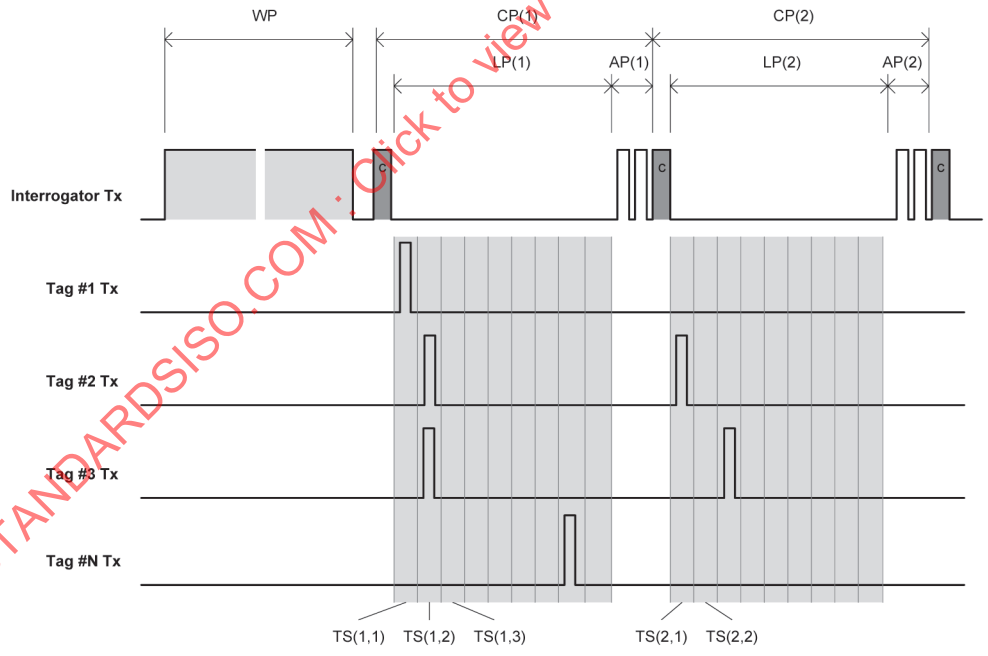


Figure 5 — Interrogator-tag communication timing diagram

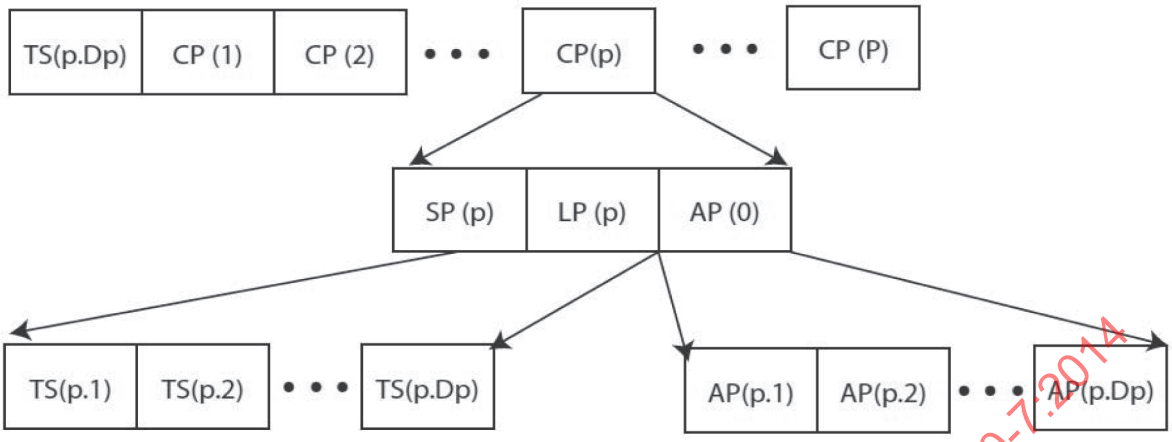


Figure 6 — Detailed Interrogator-tag anti-collision scheme timing diagram

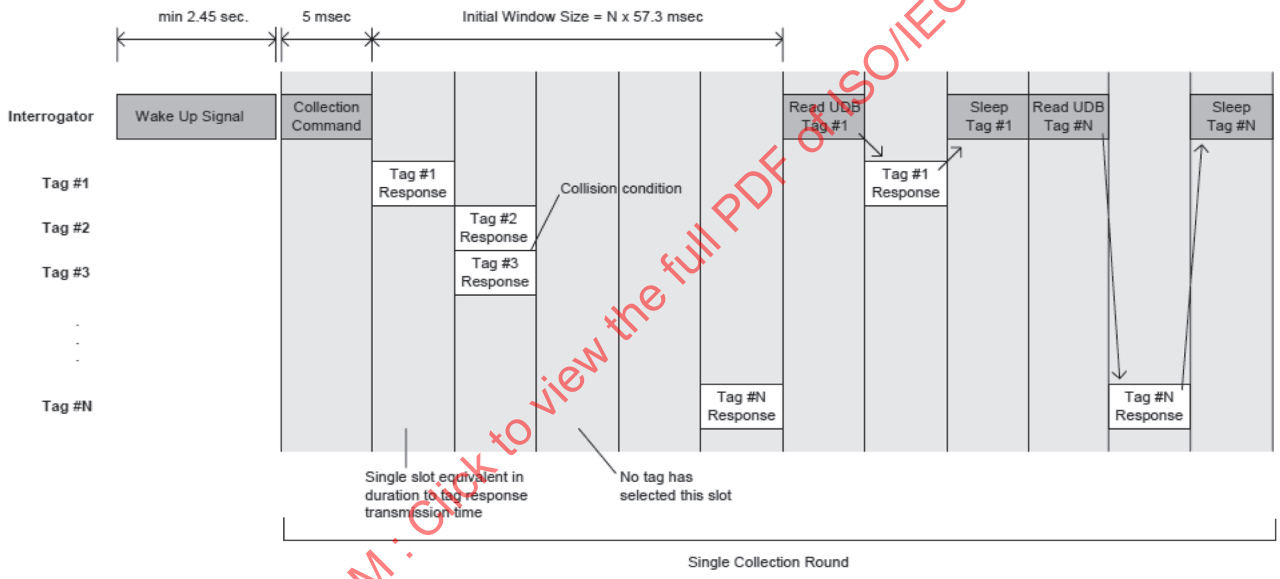


Figure 7 — Collection sequence and timing

6.5 Multi-packet UDB Collection

The following section provides a simple example of a multi-packet UDB Collection. Typically an interrogator will initiate the process by transmitting a broadcast Collect with UDB command. Tags that receive the Collect command will reply with a response packet that includes their tag identification and the first portion of the requested UDB type. The interrogator can then retrieve the remaining UDB data by sending a point-to-point Read UDB command to each tag identified in the first phase.

1. Figure 8 shows the interrogator's Collection with UDB command packet. This is a broadcast packet and all tags that receive the packet will participate in the collection process.

Protocol Code	Packet Options	Packet Length	Session ID	Command Code	Command Arguments	CRC
0x40	8 bits	1 byte	2 bytes	0x1F	4 bytes	2 bytes

Command Arguments		
Window Size	Max Packet Length	UDB Type Code
2 bytes	1 byte	1 byte

Figure 8 — Interrogator Collect with UDB broadcast for the initial collection of the tags

2. Figure 9 illustrates a tag response packet to the broadcast Collection with UDB command (command code 0x1F). The reply is in the format of a broadcast response packet.

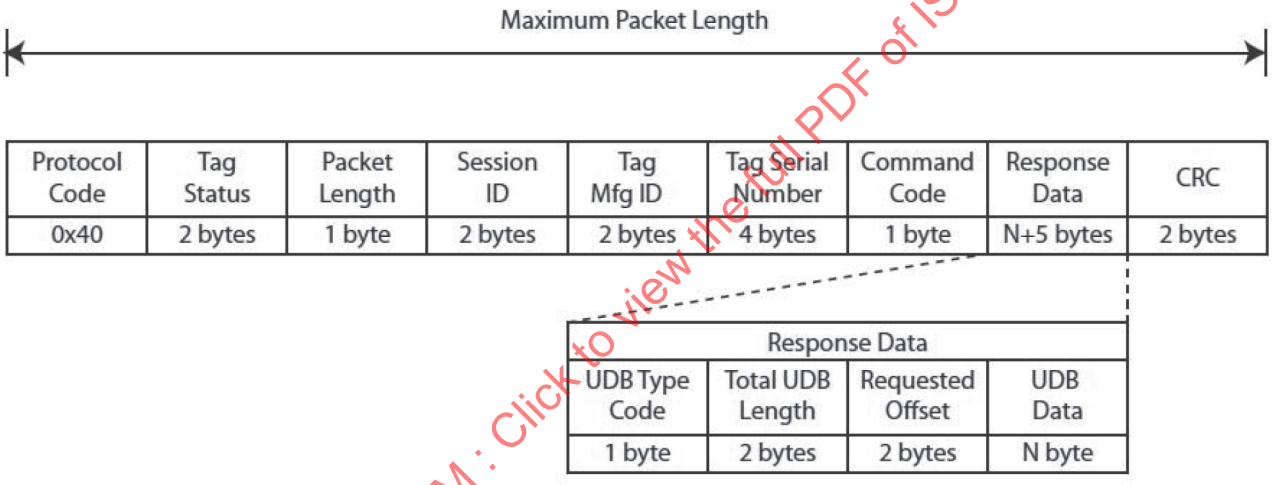


Figure 9 — Tag response to interrogator's Collection with UDB command

3. Figure 10 shows the interrogator's Read Universal Data Block command directed to a tag identified during the previous collection. The point-to-point command is directed to a specific tag using the tag identification value discovered in the previous collection. Note that the Offset into UDB field will contain a value equal to the number of bytes of UDB data returned in the Tag's collection response (N in Figure 9, above).

Protocol Code	Packet Options	Packet Length	Tag Mfg ID	Tag Serial Number	Session ID	Command Code	Command Arguments	CRC
0x40	1 byte	1 byte	2 bytes	4 bytes	2 bytes	0x70	4 bytes	2 bytes

Command Arguments		
UDB Type Code	Offset into UDB	Max Packet Length
1 byte	2 bytes	1 byte

Figure 10 — Interrogator's point-to-point request to a specific tag for the remainder of the UDB data

4. Figure 11 shows the tag reply in response to the interrogator's Read UDB command. The reply contains the second part of the UDB message using the point-to-point (directed) packet format.

Maximum Packet Length

Protocol Code	Tag Status	Packet Length	Session ID	Tag Mfg ID	Tag Serial Number	Command Code	Response Data	CRC
0x40	2 bytes	1 byte	2 bytes	2 bytes	4 bytes	1 byte	N+5 bytes	2 bytes

Response Data			
UDB Type Code	Total UDB Length	Requested Offset	UDB Data
1 byte	2 bytes	2 bytes	N bytes

Figure 11 — Tag response to interrogator's request for additional information with a point–to-point Read UDB command

6.6 Physical and Media Access Control (MAC) parameters

6.6.1 Interrogator to tag link

The interrogator-to-tag link parameters are summarized in Table 117.

Table 117 — Interrogator to tag link parameters

Ref.	Parameter	Value
Int:1	Nominal Operating Frequency range	433,92 MHz
Int:1a	Default Operating Frequency	433,92 MHz
Int:1b	Operating Channels	Not applicable in this mode
Int:1c	Interrogator Transmit Centre Frequency	±30 ppm
Int:1d	Frequency Hop Rate	Not applicable in this mode
Int:1e	Frequency Hop Sequence	Not applicable in this mode

Table 117 *(continued)*

Ref.	Parameter	Value
Int:2	Maximum Interrogator Transmit Modulation Bandwidth @-10dBc	200 kHz (transmit)
Int:2a	Minimum Tag Receiver Bandwidth @ -3dB	300 kHz
Int:3	Interrogator Transmit Maximum EIRP	As allowed by local regulations
Int:4	Interrogator Transmit Spurious Emissions	
Int:4a	Interrogator Transmit Spurious Emissions, In-Band	Not applicable in this mode
Int:4b	Interrogator Transmit Spurious Emissions, Out of Band	The interrogator shall transmit in conformance with spurious emissions requirements defined by the country's regulatory authority within which the system is operated.
Int:5	Interrogator Transmitter Spectrum Mask	Not applicable in this mode
Int:6	Timing	
Int:6a	Transmit to Receive Turn Around Time	1 ms
Int:6b	Receive to Transmit Turn Around Time	1 ms
Int:6c	Interrogator Transmit Power On Ramp	1 ms
Int:6d	Interrogator Transmit Power Down Ramp	1 ms
Int:7	Modulation	Frequency Shift Keying (FSK)
Int:7a	Spreading Sequence	Not applicable in this mode
Int:7b	Chip Rate	Not applicable in this mode
Int:7c	Chip Rate Accuracy	Not applicable in this mode
Int:7d	Modulation Index	Not applicable in this mode
Int:7e	Duty Cycle	Not applicable in this mode
Int:7f	Interrogator Transmit Frequency Deviation	±50 kHz±10kHz
Int:8	Data Coding	Manchester, 36 µs bit period. Logic one: 18 µs low followed by 18 µs high, logic zero: 18 µs high followed by 18 µs low
Int:9	Nominal Interrogator Data Bit Rate	27,778 kbit/s
Int:9a	Bit Rate Accuracy	27,778 kbit/s ±2%
Int:10	Interrogator Transmit Modulation Accuracy	Not applicable in this mode
Int:11	Preamble	
Int:11a	Preamble Length	1308 µs
Int:11b	Preamble Waveform	Square wave as defined in Int:11c
Int:11c	Bit Sync Sequence	20 cycles of 30 µs high, 30 µs low, followed by one cycle 54 µs high, 54 µs low
Int:11d	Frame Sync Sequence	20 cycles of 30 µs high, 30 µs low, followed by one cycle 54 µs high, 54 µs low
Int:12	Scrambling	Not applicable in this mode
Int:13	Bit Transmission Order	Byte: least significant bit (LSB) first Data: most significant byte first

Table 117 *(continued)*

Ref.	Parameter	Value
Int:14	Wake-up Process	Yes
Int:15	Polarization	Not Specified

6.6.2 Tag to interrogator link

The tag to interrogator link parameters are summarized in Table 118.

Table 118 — Tag to interrogator link parameters

Ref.	Parameter	Value
Tag:1	Nominal Operating Frequency range	433,92 MHz
Tag:1a	Default Operating Frequency	433,92 MHz
Tag:1b	Operating Channels	Not applicable in this mode
Tag:1c	Tag Transmit Centre Frequency	±30 ppm
Tag:1d	Frequency Hop Rate	Not applicable in this mode
Tag:1e	Frequency Hop Sequence	Not applicable in this mode
Tag:2	Maximum Tag Transmitter Modulation Bandwidth @ -10dBc	200 kHz (transmit)
Tag:2a	Minimum Interrogator Receiver Bandwidth @ -3dB	300 kHz
Tag:3	Transmit Maximum EIRP	Approximately 1mW EIRP as allowed by local regulations
Tag:4	Transmit Spurious Emissions	
Tag:4a	Transmit Spurious Emissions, In-Band	Not applicable in this mode
Tag:4b	Transmit Spurious Emissions, Out of Band	The tag shall transmit in conformance with spurious emissions requirements defined by the country's regulatory authority within which the system is operated.
Tag:5	Transmit Spectrum Mask	Not applicable in this mode
Tag:6	Timing	
Tag:6a	Transmit to Receive Turn Around Time	1 ms
Tag:6b	Receive to Transmit Turn Around Time	1 ms
Tag:6c	Transmit Power On Ramp	1 ms
Tag:6d	Transmit Power Down Ramp	1 ms
Tag:7	Modulation	Frequency Shift Keying (FSK)
Tag:7a	Spreading Sequence	Not applicable in this mode
Tag:7b	Chip Rate	Not applicable in this mode
Tag:7c	Chip Rate Accuracy	Not applicable in this mode
Tag:7d	On-Off Ratio	Not applicable in this mode
Tag:7e	Sub-carrier Frequency	Not applicable in this mode
Tag:7f	Sub-carrier Frequency Accuracy	Not applicable in this mode
Tag:7g	Sub-carrier Modulation	Not applicable in this mode
Tag:7h	Duty Cycle	Not applicable in this mode
Tag:7i	Tag Transmit Frequency Deviation	±50 kHz±10 kHz (FSK deviation)

Table 118 *(continued)*

Ref.	Parameter	Value
Tag:8	Data Coding	Manchester, 36 µs bit period. Logic one: 18 µs low followed by 18 µs high, Logic zero: 18 µs high followed by 18 µs low
Tag:9	Nominal Tag Data Bit Rate	27,778 kbit/s
Tag:9a	Bit Rate Accuracy	27,778 kbit/s ±5%
Tag:10	Tag Transmit Modulation Accuracy	Not applicable in this mode
Tag:11	Preamble	
Tag:11a	Preamble Length	1296 µs
Tag:11b	Preamble Waveform	Square wave as defined in Tag:11c
Tag:11c	Bit Sync Sequence	20 cycles of 30 µs high, 30 µs low, followed by one cycle 42 µs high, 54 µs low
Tag:11d	Frame Sync Sequence	20 cycles of 30 µs high, 30 µs low, followed by one cycle 42 µs high, 54 µs low
Tag:12	Scrambling	Not applicable in this mode
Tag:13	Bit Transmission Order	Byte: least significant bit (LSB) first Data: most significant byte first
Tag:14	(Reserved by committee)	
Tag:15	Polarization	Not Specified
Tag:16	Minimum Tag Receiver Bandwidth	200 kHz

6.6.3 Protocol parameters

The protocol parameters are summarized in Table 119.

Table 119 — Protocol parameters

Ref.	Parameter Name	Description
P:1	Who talks first	Reader-Talks-First (RTF)
P:2	Tag addressing capability	Yes
P:3	Tag UID	Yes
P:3a	UID Length	48 bit
P:3b	UID Format	binary
P:4	Read size	1-255 bytes
P:5	Write Size	1-255 bytes
P:6	Read Transaction Time	Nbytes_Read*324 µs+margin
P:7	Write Transaction Time	Nbytes_Read*324 µs+margin
P:8	Error detection	CCITT 16
P:9	Error correction	None
P:11	Command structure and extensibility	8 bits for command

6.6.4 Anti-collision parameters

The anti-collision parameters are summarized in Table 120.

Table 120 — Anti-collision parameters

Ref.	Parameter Name	Description
A:1	Type	Probabilistic
A:2	Linearity (for N tags)	Probabilistic: 0,065*N seconds for 1 <= N <= 3000
A:3	Tag inventory capacity	Probabilistic: 3000

6.7 Security architecture

This section defines a security architecture providing data and communication security including authentication and encryption between tag and interrogator. This is distinct from the password based write protection described in Section 6.3.4.

Implementation of security features such as authentication and/or encryption are optional for interrogator and tag. If authentication and/or encryption features are implemented on interrogator or tag, the implementation should conform to the security architecture defined in this section.

6.7.1 Mutual Authentication

Mutual authentication is the process by which devices in a communications link authenticate their identities to each other by sending certificates, encryption keys, and challenge information.

Two types of authentication are defined using either a pre-shared key or public keys. Multiple optional methods may be defined using the same framework. Mutual authentication with a pre-shared key defines a mandatory method with AES and SHA, while HB2-128 is provided as an optional method. Each mutual authentication method is assigned an Authentication Type Code as shown in Table 121.

Table 121 — Authentication type codes

Authentication Type Code	Mutual Authentication Description	Mandatory/Optional
0x00	RESERVED	
0x01	Pre-shared key with AES-128 and SHA-1	Mandatory
0x02	Pre-shared key with HB2-128	Optional
0x03 – 0x0F	RESERVED	
0x10	Mutual Authentication using PKI	Optional
0x11-0x7F	RESERVED	
0x80-0xFF	User defined authentication methods	Optional

6.7.1.1 Mutual authentication framework using a pre-shared Key

The framework for mutual authentication using a pre-shared key consists of two commands that support challenges and responses from both the interrogator and the tag plus any additional information required by a specific method. Optionally, a third command to distribute keys for broadcast messages is provided. The process for this mutual authentication method with a pre-shared key is shown in Figure 12 and described below.

One or more pre-shared keys are stored on each tag during a commissioning or setup process. Each pre-shared key consists of a PMK and a corresponding Key ID which serves as a nickname or reference for the PMK.

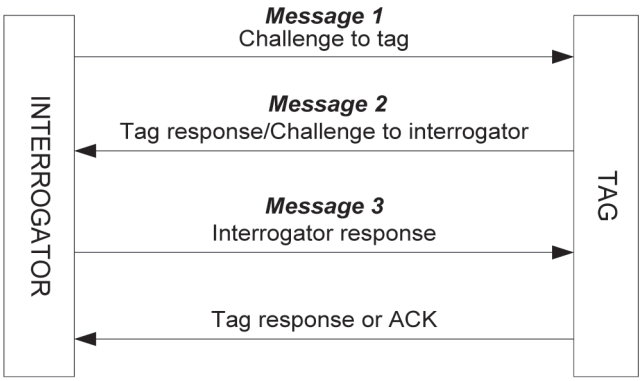


Figure 12 — Mutual authentication using a pre-shared key

This framework allows either the interrogator or the tag to initiate the first challenge and the response and a subsequent challenge may be issued within the same message. Typically, the messaging proceeds as follows:

In ***Message 1*** the interrogator initiates the mutual authentication and may send challenge parameters to the tag asking the tag to prove it knows the pre-shared key.

In ***Message 2***, if a challenge was received in Message 1 the tag typically responds to the challenge to prove it knows the pre-shared key. It may also include its challenge parameters to the interrogator within this message. If Message 2 includes the tags challenge response the interrogator typically will verify the response to authenticate the tag.

In ***Message 3*** the interrogator proves to the tag that it knows the pre-shared key by sending its response to a challenge received in Message 2. The interrogator may also send additional parameters in this message such as a broadcast key to be used only with broadcast messages. When the tag receives this message it will typically verify the interrogator's response to authenticate the interrogator.

In ***Message 4*** the tag may simply acknowledge receipt of any parameters from Message 3 or it may return a response to prove it knows the pre-shared key if not done previously. Regardless of whether the interrogator or tag initiated the first challenge, after Message 4 both the tag and the interrogator will have proved that each knows the pre-shared key and, thus, they are mutually authenticated.

Support for each of the mutual authentication messages is provided by two commands that are processed in sequence. The Secure Challenge Start command shall be sent by the interrogator as the first command to initiate mutual authentication. The subsequent command sent by the interrogator shall be the Secure Challenge End command. Optionally, the Secure Broadcast Key command may be sent by the interrogator after the Secure Challenge End command, or at any time after mutual authentication has completed. Note the commands codes for mutual authentication commands are the same for all mutual authentication methods. However, the processing of the command and its parameters may vary by method.

Table 122 — Command codes for pre-shared key mutual authentication

Command code + Sub Command Code	Command name	Command type	Description
0x50	Secure Challenge Start	Point to Point	Initial command to start mutual authentication with pre-shared key

Table 122 *(continued)*

Command code + Sub Command Code	Command name	Command type	Description
0x51	Secure Challenge End	Point to Point	Terminating command of mutual authentication with pre-shared key
0x52	Secure Broadcast Key	Point to Point	Optional command to distribute the Encrypted Broadcast Msg Key

Each of the mutual authentication framework commands has the following format:

Table 123 — Mutual authentication command format

Command Code	Authentication Type	Authentication Parameters
1 byte	1 byte	N bytes

- **Command Code:** The 1 byte value defining this command as described in Table 122.
- **Authentication Type:** The mutual authentication type defined in Table 121.
- **Authentication Parameters:** Various parameters such as key identifier, challenge data, or initialization vectors as required for mutual authentication and defined by the Authentication Type.

6.7.1.2 Mutual authentication using a pre-shared key with AES and SHA

This section describes an optional method of mutual authentication between an interrogator and tag using a pre-shared key with AES and SHA. The pre-shared key is used to derive session keys.

In this method, both the interrogator and tag are mutually authenticated if they can prove to each other they know the same pre-shared key. Once mutually authenticated they can then exchange encrypted data frames. If either the interrogator or the tag does not know the pre-shared key they will not be mutually authenticated and will not be able to exchange encrypted traffic. Retries can be performed to obtain mutual authentication.

The process for this mutual authentication method is shown in Figure 13 and described below.

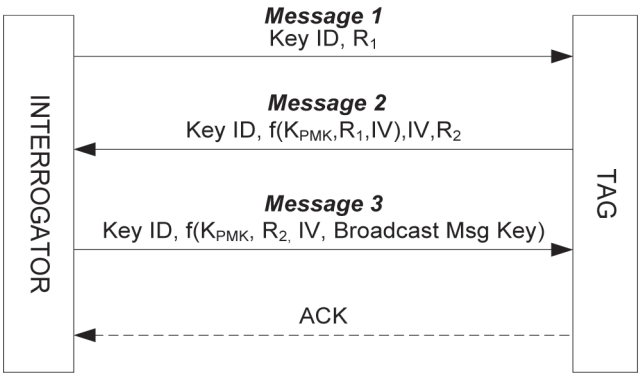


Figure 13 — Mutual authentication using a pre-shared key with AES and SHA

In ***Message 1*** the interrogator sends the challenge value $\mathbf{R_1}$ – a high quality 16 byte random number. Message 1 represents the Secure Challenge Start Command.

In ***Message 2*** the tag responds to Message 1 with $\mathbf{f(K_{PMK}, R_1, IV)}$ and $\mathbf{R_2}$. Function f(K_{PMK}, R_1, IV) presents challenge $\mathbf{R_1}$ encrypted using the pre-shared key $\mathbf{K_{PMK}}$ and **IV**. The **Initiation Vector (IV)** may

be a high quality random number 16 byte number or may start with a value of zero or a random value and be incremented by one for each additional message. It is important that the IV is unique for each encrypted message. It is better if the IV starts some value (or zero) and being incremented by one for each message.$\mathbf{K_{PMK}}$ is a 128 bit Pair-wise Master Key (PMK) derived from the pre-shared key. **IV** is the 16 byte initialization vector. $\mathbf{R_2}$ is the challenge generated by the tag – a high quality 16 byte random number. When the interrogator receives Message 2 from the tag it will decrypt the R_1. If R_1 matches the original R_1 sent by the interrogator in Message 1, then the tag proves it knows the pre-shared key and ***the tag is authenticated by the interrogator***. Message 2 represents the Secure Challenge Start Response.

In ***Message 3*** the interrogator proves to the tag that it knows the pre-shared key. It sends $\mathbf{f(K_{PMK}, R_2, IV)}$. $\mathbf{R_2}$ is the received challenge encrypted using shared key $\mathbf{K_{PMK}}$ and **IV**. When the tag receives this message it will decrypt $\mathbf{R_2}$ using its pre-shared key and compare it against the original it sent in the second message. If the decrypted and original $\mathbf{R_2}$ match then this proves the interrogator knows the same shared key and ***the interrogator is authenticated by the tag***. The interrogator and tag are now mutually authenticated. Message 3 represents the Secure Challenge End Command

Also in ***Message 3*** is the **Broadcast Msg Key** which the tag uses to encrypt and decrypt ISO/IEC 18000-7:2009 broadcast messages after authentication. The Broadcast Msg Key is encrypted using the session key $\mathbf{K_S}$ which is derived from the pre-shared key and the challenges $\mathbf{R_1}$ and $\mathbf{R_2}$. Refer to Annex B for the method to derive session keys. The session key $\mathbf{K_S}$ is used to encrypt and decrypt broadcast messages of clauses 6.2, 6.3, 6.4 and 6.5 after authentication (See e.g. Table 4).

NOTE The ACK message is shown in Figure 13. This is not part of the mutual authentication process, but is the tag response the command was received as required by this standard"

Furthermore, in Figure 13 to replace "ACK" on the fourth line with "Tag Response AES in CBC mode is used as a symmetric cryptographic algorithm for mutual authentication based on a pre-shared key.

6.7.1.2.1 Secure Challenge Start command

The Secure Challenge point-to-point command initiates mutual authentications between an interrogator and tag using AES and SHA. This command and response represent the first and second messages of mutual authentication process using a pre-shared key as shown in Figure 13.

Table 124 — Secure Challenge Start command format

Command code	Authentication Type	Key ID	Challenge R_1
0x50	0x01	2 bytes	16 bytes

- **Authentication Type:** Pre-shared key with AES and SHA.
- **Key ID:** Interrogator can propose the Key ID for the pre-shared key K_{PMK} to be used for mutual authentication. If the tag has the PMK with the same Key ID, the tag will use the PMK for mutual authentication. If the tag does not have the proposed PMK it will use one which it has and return that key ID in the second message.
- **Challenge $\mathbf{R_1}$:** high quality random 16 byte number

The tag's response includes the encrypted R_1 challenge response plus its R_2 challenge back to the interrogator. The tag shall respond to the Secure Challenge command with a point-to-point response as shown in Table 125.

Table 125 — Secure Challenge Start response format

Command code	Authentication Type	Key ID	Encrypted Challenge R_1	IV	Challenge R_2
0x50	0x01	2 bytes	16 bytes	16 bytes	16 bytes

- **Authentication Type:** Pre-shared key with AES and SHA (the same value that was sent with the command)
- **Key ID:** The ID of the pre-shared key K_{PMK} that was used to encrypt the challenge R_1.
- **Encrypted Challenge R_1:** f(K_{PMK}, R_1) –Tag's R_1 challenge response encrypted using shared key K_{PMK} (PMK) (16 bytes) and IV.
- **IV:** Initialization Vector created by tag
- **Challenge R_2**: a high quality 16 byte random number

The possible error responses shall be as shown in Table 126.

Table 126 — Secure Challenge Start command errors

Error Code	Error Name	Reason
0x50	Authentication failed	Authentication failure

6.7.1.2.2 Secure Challenge End command

The Secure Challenge End point-to-point command is used to complete the mutual authentication process method using AES and SHA. It shall be sent immediately after completion of a successful Secure Challenge Start command. This command represents the third message of mutual authentication process using a pre-shared key with AES and SHA, and also includes the distribution of the Encrypted Broadcast Msg Key..

Table 127 — Secure Challenge End command format

Command code	Authentication Type	Key ID	Encrypted challenge R_2	IV	Encrypted Broadcast Msg Key
0x51	0x01	2 bytes	16 bytes	16 bytes	16 bytes

- **Authentication Type:** Pre-shared key with AES and SHA.
- **Key ID:** The ID of the pre-shared key used to encrypt the challenge R_2. The interrogator shall use the same key ID, implying the same shared key is used by the tag in the previous message.
- **Encrypted challenge R_2:** The interrogator's R_2 challenge response. The **R_2** value received from the tag's response to the Secure Challenge command is encrypted using shared key K_{PMK} (PMK) (16 bytes) and IV.
- **IV:** Initialization Vector created by interrogator which is used to encrypt both R_2 and Broadcast key.
- **Encrypted Broadcast Msg Key:** A key used by tags to encrypt and decrypt broadcast messages. It is encrypted using the session key K_S.

The tag shall respond to the Secure Challenge End command with a point-to-point response as shown in Table 128.

Table 128 — Secure Challenge End response format

Command code
0x51

The possible error responses shall be as shown in Table 129.

Table 129 — Secure Challenge End command errors

Error Code	Error Name	Reason
0x50	Authentication failed	Authentication failure

6.7.1.2.3 Secure Broadcast Key command

The Secure Broadcast Key point-to-point command is used to distribute the Encrypted Broadcast Msg Key if the Encrypted Broadcast Key is to be transmitted to a tag.

Table 130 — Secure Broadcast Key command format

Command code	Authentication Type	Encrypted Broadcast Msg Key	IV
0x52	0x01	16 bytes	16 bytes

- **Authentication Type:** Pre-shared key with AES and SHA.
- **Encrypted Broadcast Msg Key:** This key is used by the tag to decrypt encrypted broadcast messages. It is 16 bytes and encrypted using the session key K_S as described in Annex B.
- **IV:** Initialization Vector created by the interrogator which is then used to encrypt the Encrypted Broadcast key.

The tag shall respond to the Secure Broadcast Key command with a point-to-point response as shown in Table 131.

Table 131 — Secure Broadcast Key response format

Command code
0x52

The possible error responses shall be as shown in Table 132.

Table 132 — Secure Broadcast Key command errors

Error Code	Error Name	Reason
0x50	Authentication failed	Authentication failure

6.7.1.3 Mutual authentication using a pre-shared key and HB2-128

This section describes an optional method of mutual authentication between an interrogator and tag using a pre-shared key and the HB2-128 algorithm. The pre-shared key is used to derive session keys. This method follows the mutual authentication framework described in section 6.7.1 and borrows some concepts from the AES and SHA mutual authentication method.

In this method, both the interrogator and tag are mutually authenticated if they can prove to each other they know the same pre-shared key. Once mutually authenticated they can then exchange encrypted data frames. If either the interrogator or the tag does not know the pre-shared key they will not be mutually authenticated and will not be able to exchange encrypted traffic. Retries can be performed to obtain mutual authentication.

The process of mutual authentication with a pre-shared key using the HB2-128 algorithm is shown in Figure 14 and described below.

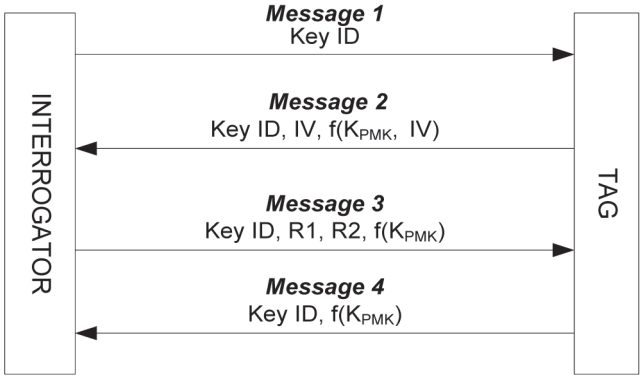


Figure 14 — Mutual authentication using a pre-shared key and HB2-128

In ***Message 1*** the interrogator sends the command to initiate the mutual authentication process. Message 1 represents the Secure Challenge Start Command.

In ***Message 2*** the tag responds to Message 1 with **f(K_{PMK}, IV)** and **IV**. Function f(K_{PMK}, IV) uses the pre-shared key **K_{PMK}** and **IV** to perform cipher initialization with IV and then performs block encryptions where the input is a function of the utilized cipher's state. **K_{PMK}** is a 256 bit Pair-wise Master Key (PMK) derived from the pre-shared key. **IV** is the 8 byte initialization vector. When the interrogator receives Message 2 from the tag it will verify the key **K_{PMK}** used by the tag is correct for that tag. If f(K_{PMK}, IV) calculated by the interrogator matches that sent by the tag, then ***the tag is authenticated by the interrogator.*** Message 2 represents the Secure Challenge Start Response.

In ***Message 3*** the interrogator proves to the tag that it knows the pre-shared key. It sends **f(K_{PMK})**. Function f(K_{PMK}) uses the pre-shared key **K_{PMK}** and corresponds to two encryptions of internal state after the 2 byte numbers **R_1** and **R_2** have been encrypted. When the tag receives this message it will encrypt **R_1** and **R_2** using its pre-shared key and current cipher internal state, then perform two block encryptions of internal state and compare the final two blocks of cipher text against the cipher text received in this message. If the final two blocks of cipher text match, then this proves the interrogator knows the same shared key and ***the interrogator is authenticated by the tag***. The interrogator and tag are now mutually authenticated for the case of the IV being generated from the current wall-clock time. Message 3 represents the Secure Challenge End Command

In ***Message 4*** the tag proves to the interrogator that it knows the pre-shared key. It sends **f(K_{PMK})**. Function f(K_{PMK}) uses the pre-shared key **K_{PMK}** and corresponds to two encryptions of internal state. When the interrogator receives this message it will encrypt two blocks beginning with its current internal state and encrypting a function of its internal state. The interrogator then compares its calculated cipher text with the two blocks sent by the tag. If the two blocks match, then this proves the tag knows the same shared key and ***the tag is authenticated by the interrogator.*** Message 4 represents the Secure Challenge End Response.

HB2-128 is used as a symmetric cryptographic algorithm for mutual authentication based on a pre-shared key.

6.7.1.3.1 Secure Challenge Start Command

The Secure Challenge point-to-point command initiates mutual authentications between an interrogator and tag. This command and response represent the first and second messages of mutual authentication process using a pre-shared key and HB2-128 as shown in Figure 14.

Table 133 — Secure Challenge Start command format

Command code	Authentication Type	Key ID
0x50	0x02	2 bytes

- **Authentication Type:** re-shared key with HB-128.
- **Key ID:** Interrogator can propose the Key ID for the pre-shared key K_{PMK} to be used for mutual authentication. If the tag has the PMK with the same Key ID, the tag will use the PMK for mutual authentication. If the tag does not have the proposed PMK it will use one that it has and return that key ID in the second message.

The tag's response includes the initialization vector IV, a high-quality 8 byte random number, and blocks of cipher text calculated by encrypting a function of internal state after the initialization process.

Table 134 — Secure Challenge Start response format

Command code	Authentication Type	Key ID	Encrypted Data	IV
0x50	1 byte	2 bytes	6 bytes	8 bytes

- **Authentication Type:** Pre-shared key with HB2-128 (the same value that was sent with the command)
- **Key ID:** The ID of the pre-shared key K_{PMK} that was used to generate the response.
- **Encrypted ID:** f(K_{PMK}, IV) –Tag's challenge of encrypted state using shared key K_{PMK} (PMK) (32 bytes) and IV.
- **IV:** Initialization Vector created by tag

The possible error responses shall be as shown in Table 135.

Table 135 — Secure Challenge Start command errors

Error Code	Error Name	Reason
0x50	Authentication failed	Authentication failure

6.7.1.3.2 Secure Challenge End Command

The Secure Challenge End point-to-point command is used to complete the mutual authentication process. It shall be sent immediately after completion of a successful Secure Challenge Start command. This command represents the third message of mutual authentication process using a pre-shared key with HB2-128, and also includes the distribution of the Encrypted Broadcast Msg Key.

Table 136 — Secure Challenge End command format

Command code	Authentication Type	Key ID	Encrypted challenge	Challenge	Encrypted Broadcast Msg Key
0x51	0x02	2 bytes	4 bytes	4 bytes	0 or 32 bytes

- **Authentication Type:** Pre-shared key with HB2-128.
- **Key ID:** The ID of the pre-shared key used to encrypt the challenge. The interrogator shall use the same key ID, implying the same shared key is used by the tag in the previous message. NOTE: The Key ID concept is the borrowed from the AES and SHA mutual authentication method.
- **Encrypted challenge:** The interrogator's challenge response to the tag. The **Challenge** value is encrypted using shared key K_{PMK} (PMK) (32 bytes) and IV.
- **Challenge:** Challenge created by interrogator.

- **Encrypted Broadcast Msg Key:** A key used by tags to encrypt and decrypt broadcast messages. It is encrypted using the session key K_S. NOTE: The Encrypted Broadcast Msg Key concept is the borrowed from the AES and SHA mutual authentication method.

The tag shall respond to the Secure Challenge End command with a point-to-point response as shown in Table 137.

Table 137 — Secure Challenge End response format

Command code	Authentication Type	Key ID	Encrypted Data
0x51	0x02	2 bytes	4 bytes

- **Authentication Type:** Pre-shared key with HB2-128 (the same value that was sent with the command)
- **Key ID:** The ID of the pre-shared key K_{PMK} that was used to generate the response. NOTE: The Key ID concept is the borrowed from the AES and SHA mutual authentication method.
- **Encrypted Data:** f(K_{PMK}) –Tag's response of encrypted state using shared key K_{PMK} (PMK) (32 bytes).

The possible error responses shall be as shown in Table 138.

Table 138 — Secure Challenge End command errors

Error Code	Error Name	Reason
0x50	Authentication failed	Authentication failure

6.7.1.3.3 Secure Broadcast Key command

The Secure Broadcast Key point-to-point command is used to distribute the Encrypted Broadcast Msg Key if the Encrypted Broadcast Key is to be transmitted to a tag.

NOTE The Secure Broadcast Key command is as defined from the AES and SHA mutual authentication method.

Table 139 — Secure Broadcast Key command format

Command code	Authentication Type	Encrypted Broadcast Msg Key	IV
0x52	0x02	16 bytes	16 bytes

- **Authentication Type:** Pre-shared key with HB2-128.
- **Encrypted Broadcast Msg Key:** This key is used by the tag to decrypt encrypted broadcast messages. It is 32 bytes and encrypted using the session key K_S as described in Annex B.
- **IV:** Initialization Vector created by the interrogator that is then used to encrypt the Encrypted Broadcast key.

The tag shall respond to the Secure Broadcast Key command with a point-to-point response as shown in Table 140.

Table 140 — Secure Challenge Broadcast Key response format

Command code
0x52

The possible error responses shall be as shown in Table 141.

Table 141 — Secure Broadcast Key command errors

Error Code	Error Name	Reason
0x50	Authentication failed	Authentication failure

6.7.1.4 Mutual Authentication with pre-shared key and multiple tags

The Base Mode collection process can be modified to include mutual authentication with multiple tags and the broadcast key distribution as depicted in Figure 15.

IMPORTANT — When security is enabled no data other than identification information shall be transmitted until both an interrogator and tag have been authenticated. When using mutual authentication within a collection process the Collect with UDB command is allowed, however, UDB data SHALL NOT be provided by the tag until after mutual authentication has been completed and data can be transferred using a secure channel. The interrogator can then use the ReadUDB command(s) to obtain UDB data over the secure channel.

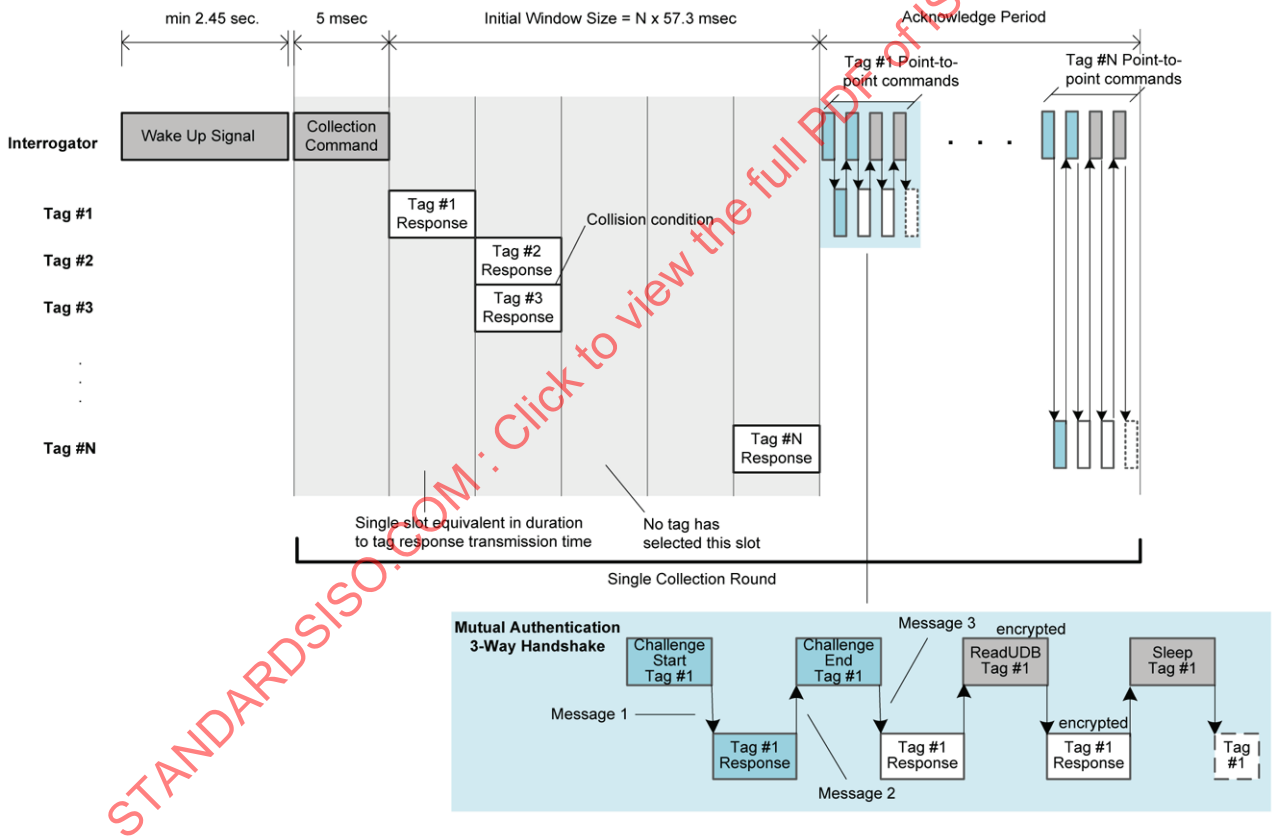


Figure 15 — Mutual Authentication/Broadcast key distribution with multiple tags

Since R_1 is a high quality 16 byte length random numbers, the R_1 sent to the Tag #1 is different from R_1 sent to Tag #2, and it is different from R_1 sent to Tag #n. The same statement is valid for all R_2s.

6.7.1.5 Mutual Authentication using PKI

This section describes a mandatory method of mutual authentication between an interrogator and tag using Public Key Infrastructure (PKI) and digital certificates. An overview of PKI and digital certificates is described in Appendix B. This method comprises a two-stage process in which digital certificates are

exchanged first, then the interrogator and tag are mutually authenticated. Once mutually authenticated they can then exchange encrypted and authenticated data frames. If either the interrogator or the tag cannot provide a signed certificate or does not have a valid private and public key pair they will not be mutually authenticated and they will not be able to exchange encrypted traffic. Retries can be performed to obtain mutual authentication.

6.7.1.5.1 Interrogator/Tag Certificate Exchange

The first stage of the process for mutual authentication with PKI is the exchange of certificates. This is shown in Figure 16 and described below.

Figure 16 — Interrogator and Tag exchange certificates

Note Signing is encrypting digest with private key - $[\,]_{\text{my-private-key}}$

Message 1 is sent by the interrogator and contains the interrogator's certificate and additional data.

— **Signing the message:** Interrogator creates a message digest using a hash function on the message. The message digest serves as a "digital fingerprint" of the message; if any part of the message is modified, the hash function returns a different result. Interrogator then *encrypts the message digest with its own private* key. This encrypted message digest is the digital signature for the message. Interrogator sends the message which includes digital signature to Tag.

— **Verifying the signature (checking the message authenticity):** When the tag receives the interrogator's certificate it decrypts the signature using the interrogator's public key, thus revealing the message digest. To verify the message, tag then hashes the message with the same hash function Interrogator used and compares the result to the message digest it received from Interrogator. If they are exactly equal, tag can be confident that the message did indeed come from the interrogator and has not been changed. The tag uses the interrogator's public key to verify the signature, so if the message digests are not equal the message either originated elsewhere or was altered after it was signed.

— **Checking the Interrogator's certificate authenticity:** Tag can check the interrogator's certificate signature using the CA public key. Tag will hash all data from the certificate to create the signature. Tag will decrypt original cert signature using CA's public key. Tag will compare these two signatures. If they are the same the interrogator certificate has not been forged. Tag saves interrogator's digital signature which contains interrogator's public key.

Message 2 is sent by the tag and contains tag's certificate and additional data. The process of creating the message, signing the message, checking the message and certificate authenticity is the same as described for Message 1 except the interrogator and tag roles are swapped.

At the end of the certificate exchanges:

- Both the interrogator and tag have verified if a received certificate is valid and has not been forged.
- Interrogator has its private and public key, its certificate signed with a private key of Certificate Authority (CA), CA certificate and Tag's certificate.
- Tag has its private and public key, its certificate signed with a private key of Certificate Authority (CA), CA certificate and Interrogator's certificate.

6.7.1.5.2 Interrogator/Tag Mutual Authentication

The second stage of the process for mutual authentication with PKI is mutual authentication with signed messages. This is shown in Figure 17 and described below.

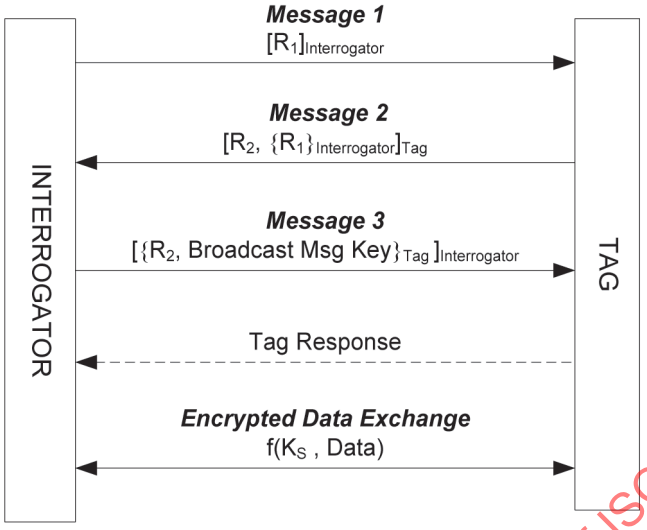


Figure 17 — Interrogator and Tag Mutual Authentication using Public Keys

NOTE Signing (encrypt digest) - $[\,]_{\text{my-private-key}}$ Encrypt - $\{\}_{\text{other-side-public-key}}$

In ***Message 1*** the interrogator sends to the tag the challenge value $\mathbf{R_1}$ – a high quality 16 byte random number. The complete message is signed using the interrogator's private key. The tag receives the message and, using the interrogator's public key received previously when certificates were exchanged, verifies the message signature.

In ***Message 2*** the tag responds to Message 1 with encrypted $\mathbf{R_1}$ using the interrogator's public key and $\mathbf{R_2}$, a random number. The complete message is signed using the tag's private key. When the interrogator receives Message 2 it verifies the message signature using the tag's public key. If the message signature is verified the interrogator then decrypts R_1 and compares it to the original R_1 sent by the interrogator in the Message 1. If they match then ***the tag is authenticated by the interrogator***.

In ***Message 3*** the interrogator responds to tag's challenge with $\mathbf{R_2}$, a random number which is encrypted using the tag's public key. The complete message is signed using the interrogator's private key. When the tag receives Message 3 it verifies the message signature using the interrogator's public key. If the message signature is verified the tag then decrypts R_2 using its private key and compares it to the original R_2 it sent in Message 2. If they match then ***the interrogator is authenticated by the tag***.

Also in **Message 3** is the Broadcast Msg Key which the tag uses to encrypt and decrypt broadcast messages after authentication. The Broadcast Msg Key is encrypted using the tag's public key.

NOTE The ACK message is shown in Figure 17. This is not part of the mutual authentication process, but is the tag response to the interrogator command as required by this standard.

Once Interrogator and Tag are mutually authenticated they can exchange as many encrypted Data messages as needed. Only authenticated tag can get the broadcast key from the interrogator.

6.7.1.5.3 Base Mode commands for mutual authentication using PKI certificates

The following describes the commands and responses required to support the mutual authentication process using PKI with digital certificates as described in clauses 6.7.1.5.1 and 6.7.1.5.2. Note these commands are optional, but if any one is supported then all shall be supported.

Table 142 — Command codes for mutual authentication using PKI

Command code + Sub Command Code	Command name	Command type	Description
0x53	PKI Cerificate Exchange	Point to Point	Exchanges interrogator and tag digital certificates for PKi
0x50	Secure Challenge Start	Point to Point	First phase of mutual authentication with pre-shared key
0x51	Secure Challenge End	Point to Point	Final phase of mutual authentication with pre-shared key
0x52	Secure Broadcast Key	Point to Point	Distributes a Broadcast Msg Key

6.7.1.5.3.1 PKI Certificate Exchange command

The PKI Certificate Exchange command is used to exchange digital certificates between an interrogator and one or more tags as the first state of mutual authentication using PKI as shown in Figure 17. This command may be sent as either a broadcast or point-to-point command.

Table 143 — PKI Certificate Exchange command/response format

Command code	Authentication Type	Certificate	Message Signature
0x53	0x10	N bytes	N bytes

- **Authentication Type:** Mutual authentication based on PKI certificates
- **Certificate:** The interrogator's (for command) or tag's (for response) unique public key certificate which contains its identifying information and its public key, and is signed by the Certification Authority. See Appendix B for information on public key certificates.
- **Message Signature**: The message digest encrypted with the interrogator's (for command) or tag's (for response) private key.

When the tag receives the interrogator's certificate it first extracts the interrogator's public key from the certificate. With this public key it then decrypts the message signature to reveal the message digest. To verify the message, tag then hashes the message with the same hash function the interrogator used and compares the result to the message digest received from interrogator. If they are exactly equal, the tag can be confident the message is authentic. If the message is authentic then the tag shall respond as shown in Table 143 except that the certificate and message signature are from the tag instead of the interrogator.

The possible error responses shall be as shown in Table 144.

Table 144 — PKI Certificate Exchange command errors

Error Code	Error Name	Reason
0x50	Authentication failed	Authentication failure

6.7.1.5.3.2 Secure Challenge Start command

The Secure Challenge point-to-point command initiates mutual authentications between an interrogator and tag. This command and response represent the first and second messages of mutual authentication process using PKI as shown in Figure 17.

Table 145 — Secure Challenge Start command format

Command code	Authentication Type	Challenge R_1	Message Signature
0x50	0x10	16 bytes	N bytes

- **Authentication Type:** Mutual authentication based on PKI certificates
- **Challenge R_1:** high quality random 16 byte number
- **Message Signature**: The message digest encrypted with the interrogator's private key.

The tag's response includes its encrypted R_1 challenge response plus its R_2 challenge back to the interrogator. The tag shall respond to the Secure Challenge command with a point-to-point response as shown in Table 146.

Table 146 — Secure Challenge Start interrogator response format

Command code	Authentication Type	Encrypted challenge R_1	Challenge R_2	Message Signature
0x50	0x10	16 bytes	16 bytes	N bytes

- **Authentication Type:** Mutual authentication based on PKI certificates
- **Encrypted challenge R_1:** The tag's R_1 challenge response encrypted using interrogator's public key
- **Challenge R_2**: R_2, a high quality 16 byte random number
- **Message Signature**: The message digest encrypted with the tag's private key.

The possible error responses shall be as shown in Table 147.

Table 147 — Secure Challenge Start command errors

Error Code	Error Name	Reason
0x50	Authentication failed	Authentication failure

6.7.1.5.3.3 Secure Challenge End command

The Secure Challenge End point-to-point command is used to complete the mutual authentication process. It shall be sent immediately after completion of a successful Secure Challenge Start command. This command represents the third message of mutual authentication process using PKI, and also includes the distribution of the Encrypted Broadcast Msg Key.

Table 148 — Secure Challenge End command format

Command code	Authentication Type	Encrypted challenge R_2	Encrypted Broadcast Msg Key	Message Signature
0x51	0x10	16 bytes	16 bytes	N bytes

- **Authentication Type:** Mutual authentication based on certificates
- **Encrypted challenge R_2:** The interrogator's R_2 challenge response encrypted using tag's public key.
- **Encrypted Broadcast Msg Key:** A key used by tags to encrypt and decrypt broadcast messages. It is encrypted using the tag's public key.

- **Message Signature**: The message digest encrypted with the interrogator's private key.

The possible error responses shall be as shown in Table 149.

Table 149 — Secure Challenge End command errors

Error Code	Error Name	Reason
0x50	Authentication failed	Authentication failure

6.7.1.5.4 Exchanging Certificates in the Base Mode Collection Procedure

The collection process can be used for implementing certificate exchange with multiple tags as depicted in the Figure 18 below. The Collect with UDB command shall be sent with a Max Packet Length of 20 (x014) so that no UDB data is returned and collect only the tag's identity. After certificate exchange, another collection process may be initiated so as to do mutual authentication and collection of UDB data. See clause 6.7.1.5.5 for details.

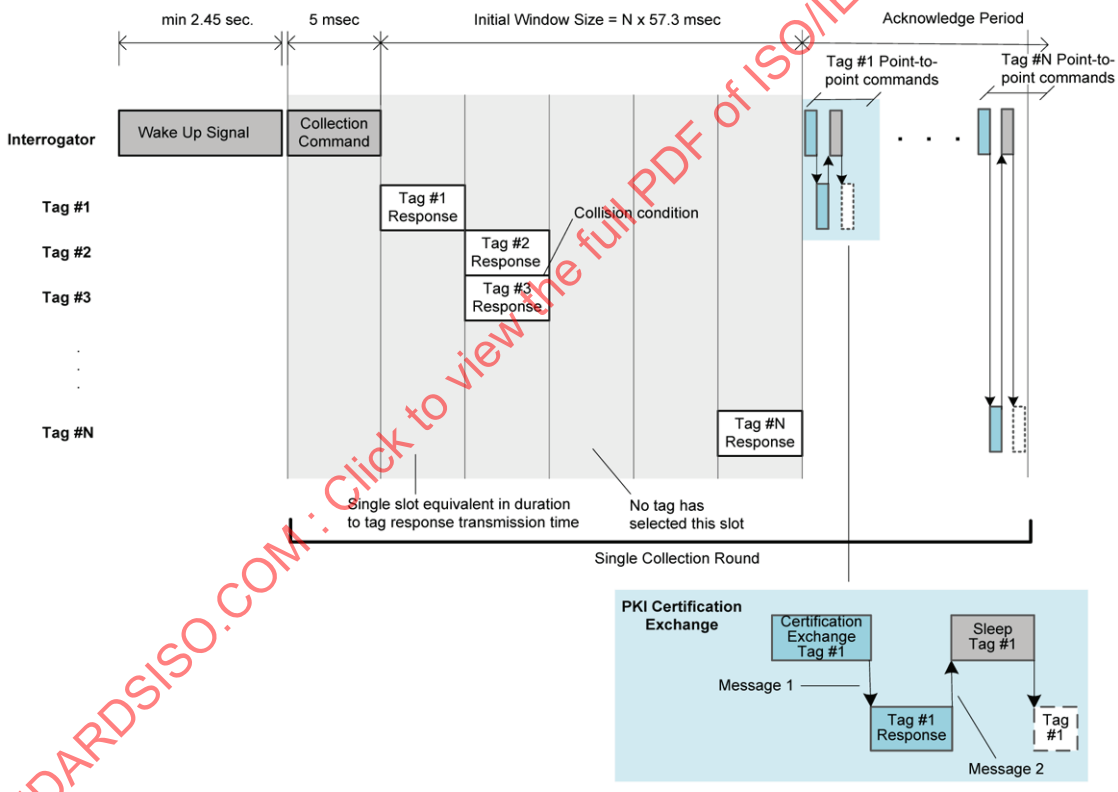


Figure 18 — Certificate Exchange sequence

6.7.1.5.5 Base Mode Extension Mutual Authentication using PKI and multiple tags

The collection process can be used for implementing Mutual Authentication using PKI with multiple tags and the broadcast key distribution as depicted in Figure 19.

IMPORTANT — When security is enabled no data other than identification information shall be transmitted until both an interrogator and tag have been authenticated. When using mutual authentication within a collection process the Collect with UDB command is allowed, however, UDB data SHALL NOT be provided by the tag until after mutual authentication has been completed and data can be transferred using a secure channel. The interrogator can then use the ReadUDB command(s) to obtain UDB data over the secure channel.

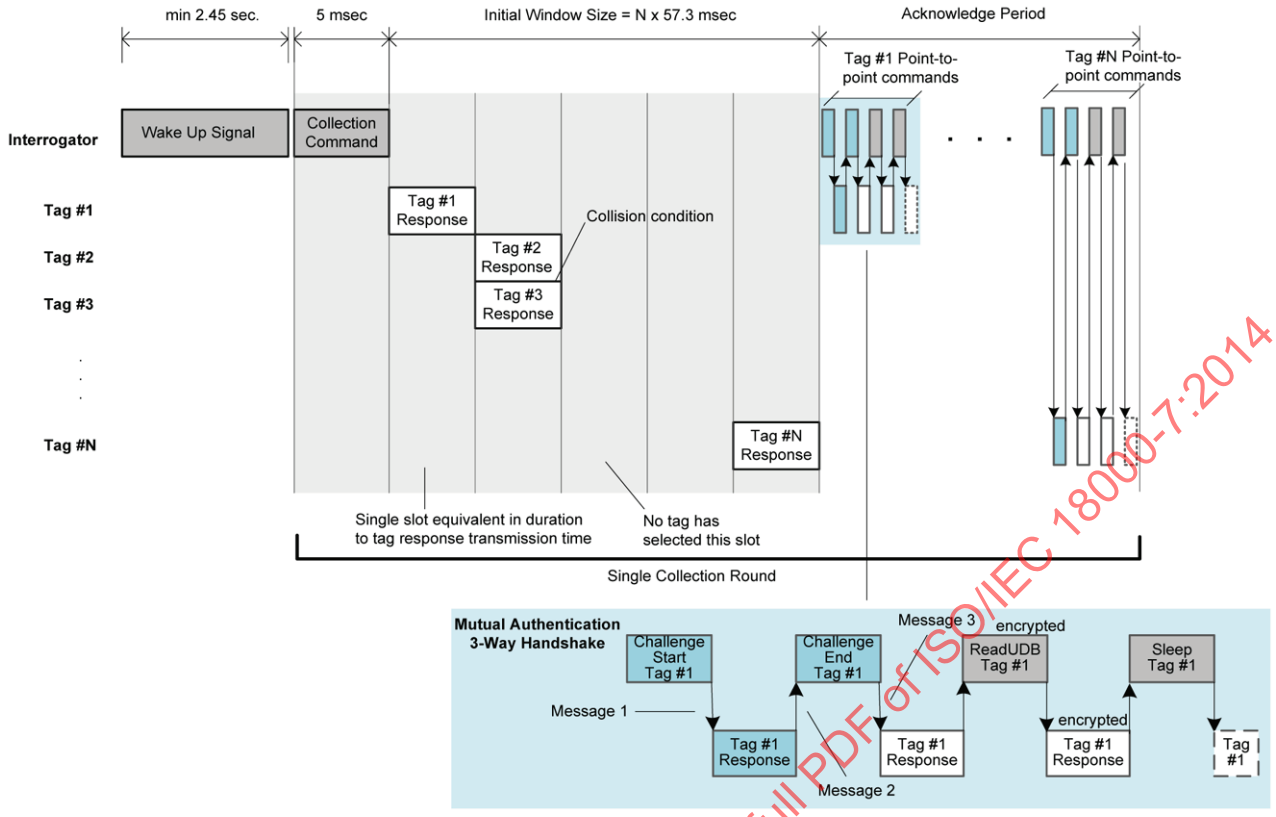


Figure 19 — Mutual Authentication with multiple tags using PKI and Broadcast key distribution

Since R_1 is a high quality 16 byte length random numbers, the R_1 sent to the Tag #1 is different from R_1 sent to Tag #2, and it is different from R_1 sent to Tag #n. The same statement is valid for all R_2s.

6.7.1.5.6 Encrypted Data Transfer

Once mutually authenticated by either the pre-shared key or PKI methods an interrogator and tag can then exchange encrypted data messages for as long as a session between them exists. Mutual authentication between an interrogator and tag may terminate when one of the following activities occur:

- An interrogator commands the tag to sleep using the Sleep or Sleep All But command.
- A tag's has not received a valid message for 30 seconds and transitions to the sleep state.
- An interrogator initiates another mutual authentication sequence.

6.7.2 Frame Security

6.7.2.1 Security protocol ID

Significant changes to the command and response packets are required to support security. Therefore, it will be necessary to create a new protocol ID (0x50) to support security and maintain compatibility and coexistence with Base Mode devices. It is assumed that devices not supporting this security extension will check the protocol ID to avoid interoperability and conformance problems.

6.7.2.2 Frame protection

This clause describes confidentiality and integrity of wireless frames. The following variants of the frame format of Base Mode are considered:

— Securing the payload of the ISO/IEC 18000-7 frames (Commands and command parameters)

For the frames described below some fields previously defined in ISO/IEC 18000-7 are reused, and some fields are new. Only the new fields are described. Refer to ISO/IEC 18000-7 for information of reused fields

6.7.2.3 Encryption/Authentication of the frame payload

Table 150 — Interrogator-to-tag frame format (broadcast)

Protocol ID	Packet Options	Packet Length	Session ID	IV/CCM Header	Encrypted Payload	Authentication Data (SHA1-96|SHA1|CBC-MAC|HB2-128)	CRC
0x50	1 byte	1 byte	2 bytes	16/8 bytes	N bytes	12/20/8 bytes	2 bytes

- **IV/CCM Header:** The initialization vector used to encrypt the frame with a symmetric algorithm
- **Encrypted Payload:** All data including the command and parameters that is encrypted.
- **Authentication Data:** The hash function result calculated over the payload prior to encryption

Table 151 — Interrogator-to-tag command format (point-to-point)

Protocol ID	Packet Options	Packet Length	Tag Manufacturer ID	Tag Serial Number	Session ID	
0x50	1 byte	1 byte	2 bytes	4 bytes	2 bytes	...

...	IV/CCM Header	Encrypted Payload	Authentication Data (SHA1-96|SHA1|CBC-MAC|HB2-128)	CRC
	16/8 bytes	N bytes	12/20/8 bytes	2 bytes

- **IV/CCM Header:** The initialization vector used to encrypt the frame with a symmetric algorithm
- **Encrypted Payload:** All data including the command and parameters that is encrypted.
- **Authentication Data:** The hash function result calculated over the payload prior to encryption

Table 152 — Tag-to-interrogator response format (broadcast and point-to-point)

Protocol ID	Tag Status	Packet Length	Session ID	Tag Manufacturer ID	Tag Serial Number	
0x50	1 byte	1 byte	2 bytes	2 bytes	4 bytes	...

...	IV/CCM Header	Encrypted Payload	Authentication Data (SHA1-96|SHA1|CBC-MAC|HB2-128)	CRC
	16/8 bytes	N bytes	12/20/8 bytes	2 bytes

- **IV/CCM Header:** The initialization vector used to encrypt the frame with a symmetric algorithm
- **Encrypted Payload:** All data including the command and parameters that is encrypted.
- **Authentication Data:** The hash function result calculated over the payload prior to encryption

This frame format provides privacy and authentication of the payload, but does not include protection for the tag and interrogator identity information such as Tag Manufacturer ID, Tag Serial Number, and Session ID. In this case it is possible to have a unique key per tag. Tag Manufacturer ID, Tag Serial Number, and Tag Manufacturer ID shall be used to determine the tag's key.

Tag Manufacturer ID, Tag Serial Number, and Session ID are sent in the clear.

The size of the Authentication field depends on the selected authentication algorithms. The replay protection can be done by enforcing IV sequencing.

6.7.2.4 Packet Options

Table 153 — Packet options field

BITS							
7 **PKI** *See* 7.3.6.1.4	6	5	4 **Protection suites** *See* Table 187	3 **Secure**	2	1	0
0=PKI not used 1=PKI used				0= Frame is not encrypted (there is no IV and authentication field in the frame) 1= Frame is encrypted	1	0= Broadcast (Tag serial number and Tag manufacturer ID not present) 1= Point to Point (Tag serial number and tag manufacturer ID present)	Reserved

The Packet Options field Bit 3, described in the above table, shall be used to indicate if the frame is secured or not. If the value of the bit 3 is:

- If Secure is set to 1 then the frame is secured and the fields IV/CCM Header, Encrypted Payload, and Authentication Data are present in the frame. The command code and command arguments are encrypted within the Encrypted Payload field.
- If Secure is set to 0 then the frame is NOT secured and the fields IV/CCM Header, Encrypted Payload, and Authentication Data are NOT present in the frame. The command code and command arguments are NOT encrypted.

6.7.2.5 Protection suites – Symmetric cryptography

Cryptographic protection of Base Mode messages consists of an optional message encryption step followed by a message authentication step. Each protection suite type is assigned an enumerated descriptor also called a protection suite identifier. A protection suite identifier is written as a designator of an encryption method followed by a designator of an authentication method:

<message-encryption-method>-<message-authentication-method>

The protection suite descriptor uniquely identifies the methods used to authenticate and encrypt the message.

Table 154 specifies crypto protection suites.

Table 154 — Base Mode Protection Suites

Protection Suite Descriptor	Encryption Algorithm	Key Size, bit	Encryption Mode	Authentication Algorithm	Authentication size, # bits	Packet Options (Protection suites bits)
AES-128-CBC-SHA1-96	AES	128	CBC	SHA1-96	96	001

Table 154 *(continued)*

Protection Suite Descriptor	Encryption Algorithm	Key Size, bit	Encryption Mode	Authentication Algorithm	Authentication size, # bits	Packet Options (Protection suites bits)
AES-128-CBC-SHA1	AES	128	CBC	SHA1	160	010
CCM -7 (CCM for ISO/ IEC 18000-7)	AES	128	Counter	AES-CBC-MAC	64	011
NULL – NULL (secure bit is 0)	None	N/A	N/A	None	N/A	000
HB2-128	HB2-128	128	Normal	HB2-128	16-128	100

Packet Options (Protection suites bits) values 110, 101, 111 are for future extensions.

CCM-7 is based on CCM, a generic authenticated encryption block cipher mode of AES. CCM is a mode of operation defined for any block cipher with a 128-bit block size. CCM combines two well-known and proven cryptographic techniques to achieve robust security. First, CCM uses CTR for confidentiality and Cipher Block Chaining MAC (CBC-MAC) for both authentication and integrity protection.

6.7.2.6 Protection suites – Asymmetric cryptography

When PKI is used, PKI bit in the Packet Options is set to 1. Protection suite bits will code the PKI algorithm used for mutual authentication:

001 ECC

010 RSA

Packet Options (Protection suites bits) values 00, 011, 100, 101, 110, 111 are reserved for future extensions.

6.7.3 Tag Data Access

This section describes the methods of security to allow or prevent access to data residing on ISO/IEC 18000-7 tags.

6.7.3.1 Tag data storage overview

Data storage on a Base Mode tag is described in Table 155.

Table 155 — Data Storage Summary

Storage	Contents	Access	Access Commands
Universal Data Block (UDB)	Transit data (Type 0) • Routing Code • User ID UDB • Application data	Read Only	Read UDB R/W Routing Code R/W User ID
	Capability data (Type 1) • Optional Commands • Memory Size • Table Query Size • Application data	Read Only	Read UDB
	Query Results (Type 2) • Application data	Read Only	Read UDB
	Hardware Fault (Type 3) • Hardware Status • Application data	Read Only	Read UDB
R/W Memory	User defined data	Read/Write	Read Memory, Write Memory
Database Tables	User defined data	Read/Wriite	Table commands

7 Extended Mode

7.1 General description

An LR-WPAN is a simple, low-cost communication network that allows wireless connectivity in applications with limited power and relaxed throughput requirements. The main objectives of an LR-WPAN are ease of installation, reliable data transfer, short-range operation, extremely low cost, and a reasonable battery life, while maintaining a simple and flexible protocol. Some of the characteristics of an LR-WPAN are as follows:

— Over-the-air data rates of 250 kb/s, 100kb/s, 31.25 kb/s

— Star or peer-to-peer operation

— Allocated 8-bit short or 64-bit extended addresses

— Optional allocation of guaranteed time slots (GTSs)

— Carrier sense multiple access with collision avoidance (CSMA-CA) channel access

— Fully acknowledged protocol for transfer reliability

— Low power consumption

— Energy detection (ED)

— Link quality indication (LQI)

Four different device types can participate in an IEEE 802.15.4 network; a full-function device (FFD), a reduced-function device (RFD), reduced function receive only device (RFD-RX) and reduced function transmit only device (RFD-TX). The FFD can operate in three modes serving as a personal area network (PAN) coordinator, a coordinator, or a device. An FFD can talk to RFDs or other FFDs, while an RFD can talk only to an FFD. An RFD is intended for applications that are extremely simple, such as a light

switch or a passive infrared sensor; they do not have the need to send large amounts of data and may only associate with a single FFD at a time. Consequently, the RFD can be implemented using minimal resources and memory capacity.

In order to align Base Mode of ISO 18000-7 Accept which is specified in Clause 6, new air interface is being specified in addition to already existing Base Mode. This air interface is called Extended Mode and specifies the following:

— Physical layer (PHY), which is adopted from IEEE 802.15.4f-2012 standard

— Media access control layer (MAC) which is adopted from IEEE 802.15.4-2011 and IEEE 802.15.4e-2012 standard

— Logical Link Control layer (LLC) which provides interface between MAC and higher networking layers (e.g. IPv4)

— Wake-On mechanisms which provide methods to wake up the device from sleep mode

All specifications for PHY and MAC layers are adopted from IEEE 802.15.4-2011 standard and its addendums: IEEE 802.15.4f-2012 and IEEE 802.15.4e-2012 standards. Those documents provide complete normative specification for PHY and MAC layer of Extended Mode.

7.1.1 Architecture

Extended Mode is a layered architecture that is based on the Open System Interconnection (OSI) seven layer model defined in ISO/IEC 7498-1:1994. Each layer represents one aspect of the complete architecture and provides management and data services to the higher layers. The interface between each layer provides the logical link to these layers. Figure 20 illustrates the protocol layers supported by Extended Mode.

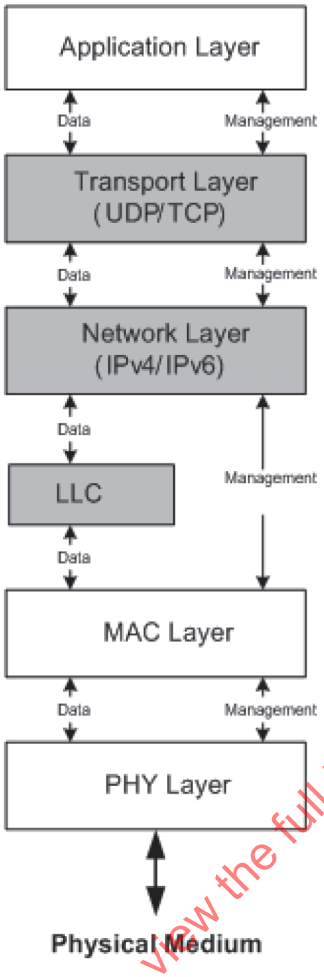


Figure 20 — Extended Mode Protocol Stack

The PHY layer is responsible for transport of bit-stream data across the radio link and management of the hardware to achieve this. This includes enabling and disabling of the radio transceiver, channel selection, clear channel detection and assessment, and collision avoidance. Additionally, the PHY provides controls for conservation of device power as determined by higher layers.

The MAC sub-layer is responsible for data and management services which include transmission and reception of packet data from the PHY and synchronization, validation, and acknowledgement of frame delivery. Additionally, the MAC sub-layer manages power savings for the device, the association and disassociation of devices within its network, and provides infrastructure for security within this layer.

The Logical Link Control (LLC) sub-layer can access the MAC sub-layer through a service-specific convergence sub-layer (SSCS). The detail of this functionality is beyond the scope of this document.

The upper layers consist of the Network layer, the Transport layer, and the Application layer. The Network layer provides configuration, manipulation, and message routing. The Transport layer controls the reliability of a given link using flow control, segmentation, and error control, and acknowledges successful data transmission to the Application layer. The Application layer provides the intended functionality of the device. The definition of the LLC, Network, and Transport layers is outside the scope of this document.

Extended Mode devices incorporate a minimum of a PHY layer, a MAC sub-layer, and an Application layer. The LLC, Network, and Transport layers are optional and may present an excessive frame size versus the short frame with direct routing to the Application layer. Still, a clear separation of protocol layers is maintained in this case.

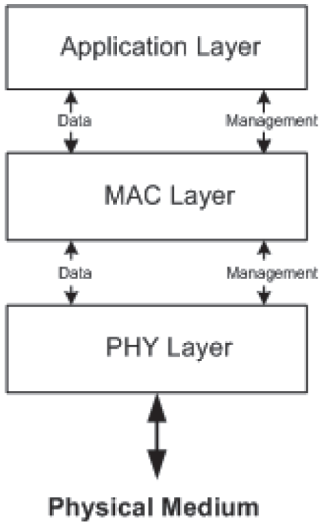


Figure 21 — Typical Extended Mode protocol stack

7.1.2 Extended mode components

Extended Mode devices are not defined strictly as Interrogators and Tags as they are in Base Mode. Instead Extended Mode Devices are defined as three standardized device settings called Endpoints, Subcontrollers, and Gateways. Other settings are possible and subject to future inclusion in this standard. Moreover, a single device is permitted to switch between multiple settings as necessary by the application.

7.1.2.1 Endpoint Setting

The Endpoint Setting is similar in behaviour to a tag in the Base Mode. While in an Endpoint Setting, an Extended Mode device spends most of its time in a low-power state. Once the device awakens (by receiving a wake-on event or triggered by internal sensor or triggered by internal timer), it shall engage in a process of processing the request and usually accessing the communication channel through one of the channel access methods defined in this specification.

7.1.2.2 Subcontroller Setting

The Subcontroller Setting behaves halfway between the Endpoint Setting and the Gateway Setting. Devices in the Subcontroller Setting open and maintain communication with Endpoint Setting devices or other Subcontroller Setting devices. Devices that support the Subcontroller Setting shall also support the Endpoint Setting.

7.1.2.3 Gateway Setting

The Gateway Setting behaves much like a typical implementation of the Base Mode's interrogator: it is always on, it is actively receiving, and it is generally connected to a wire-line power-supply. In Extended Mode, the Gateway Setting is defined in two ways:

It is a device that connects an Extended Mode network to another type of network.

It is optimized for lowest latency channel access and network arbitration.

7.2 Physical (PHY) Layer

Extended Mode specifies a physical layer that is not interoperable with the Base Mode PHY, but it is similar enough that RF resources capable of generating the Extended Mode PHY can typically generate

the Base Mode PHY as well. Additionally, Extended Mode specifies provisions for coexistence with Base Mode via configuration options that propagate through most layers of the stack.

This clause specifies the Extended Mode MAC sub-layer. It is based and fully compliant with IEEE 802.15.4f-2012 standard specification. Only some elements of IEEE 802.15.4f-2012 specification are utilized in Extended Mode as described in this clause. For full specification and implementation details refer to IEEE 802.15.4f-2012 standard specification (433 MHz portion).

7.2.1 Spectrum Utilization and Channels

ISO 18000-7 Extended Mode uses the 433 MHz ISM Band, occupying 433.05 to 434.79 MHz. The formal Extended Mode spectrum extends from 433.056 to 434.784 MHz, organized into channels. Each channel is defined by an index specifying its center frequency and an index specifying its bandwidth. Any given device may, at any given time, permit communication on any combination of supported channels. Optimal practice, however, is to minimize the usage of overlapping channels within a single network.

Table 156 specifies spectrum and channel conformance of Extended Mode to IEEE 802.15.4f-2012.

Table 156 — Extended Mode Conformance to IEEE 802.15.4f-2012 Spectrum and Channels

Clause of IEEE 802.15.4f-2012	Title	Conformance	Comment
8	General PHY Requirements	-	-
8.1.1.	Operating Frequency Range	Partial	Only 433 MHz PHY supported
8.1.2.	Channel Assigment	Yes	
8.1.2.6.	Channel numbering for MSK PHY 433 MHz band	Yes	
8.1.2.7.	Channel numbering for MSK PHY 2450 MHz band	No	Only 433 MHz PHY supported
8.1.2.8.	Channel numbering for LRP UWB PHY	No	Only 433 MHz PHY supported

7.2.1.1 Channel Center Frequencies

Channel center frequencies are indexed pursuant to Table 157. Channel identification includes the channel frequency index.

Table 157 — Channel Center Frequency Indexes

Center Freq. Index	Center Frequency	Center Freq. Index	Center Frequency
0x00	433.164 MHz	0x01	433.272 MHz
0x02	433.380 MHz	0x03	433.488 MHz
0x04	433.596 MHz	0x05	433.704 MHz
0x06	433.812 MHz	0x07	433.920 MHz
0x08	434.028$_+$ MHz	0x09	434.136 MHz
0x0A	434.244 MHz	0x0B	434.352 MHz
0x0C	434.460 MHz	0x0D	434.568 MHz
0x0E	434.676 MHz	0x0F	—

7.2.2 RF Modulation

Table 158 specifies conformance of Extended Mode to IEEE 802.15.4f-2012 relevant to RF modulation.

Table 158 — Extended Mode Conformance to IEEE 802.15.4f-2012 RF Modulation

Clause of IEEE 802.15.4f-2012	Title	Conformance	Comment
16.4	MSK Modulation	Yes	
16.4.1	Reference Modulator Diagram	Yes	
16.4.2	Bit-To-Symbol Mapping	Yes	
16.4.3	Signal Modulation	Yes	
16.5	MSK PHY Requiremetns	-	
16.5.1	Operating Frequency Range	Partial	Only 433 MHz PHY supported
16.5.2	Transmit PSD Mask	Partial	Only 433 MHz PHY supported
16.5.3	Symbol Rate	Yes	
16.5.4	Transmit Center Frequency Tolerance	Yes	
16.5.5.	Transmit Power	Partial	Only 433 MHz PHY supported
16.5.6.	Receiver maximum input level of desired signal	Yes	
16.5.7.	Modulation frequency deviation tolerance	Yes	
16.5.8	Zero Crossing Tollerance	Yes	

7.2.2.1 Channel Bandwidths

Channel bandwidths are indexed pursuant to Table 159. Channel identification includes the channel bandwidth index. The Channel Bandwidth Index also stipulates the type of modulation and symbol rate the identified channel shall utilize.

Table 159 — Channel Bandwidth Indexes

Bandwidth Index	Channel Bandwidth	Modulation	Symbol Rate
0x00	0.324 MHz	MSK (Normal)	100 ksymbols/s
0x01	0.108 MHz	MSK (Narrow band)	31.25 ksymbols/s
0x02	0.540 MHz	MSK (Wide band)	250 ksymbols/s

7.2.2.2 Channel Classes

Channel Classes specify the data rates, message modulation types, passband requirements, and stopband requirements for each Extended Mode channel. Channels in a given class may also be required to participate in a CSMA process prior to transmission.

Normal Class — Basic channel class.

Wide-Band Class — Maximum data rate

Narrow-Band Class — Low data rate

Tables 160 and 161 lists channel parameters which are more stringent than channel parameters specified in IEEE 802.15.4f-2012 or are not specified in IEEE 802.15.4f but are important parameters required to meet some of worldwide regulatory requirements relevant to RFID devices.

Table 160 — Basic Physical Requirements for all Modulation Classes

Specification	Min	Typ	Max	Units
Peak to Channel-Stopband Attenuation	30			dB
EIRP at Channel-Stopband	-40			dBm
EIRP at Neighboring Channel-Stopband	-60			dBm
Passband EIRP		0		dBm

Table 161 — Additional Physical Requirements per Modulation Class

Channel Class		CSMA	Carrier Sense Min Sensitivity	1% BERMin Sensitivity	
Normal		Optional	-110 dBm	-90 dBm	
Wide-Band		Optional	-96 dBm	-84 dBm	
Narrow-Band		Optional	-110 dBm	-92 dBm	

7.2.2.3 Wide-Band MSK Modulation

Certain channel classes use a Wide-Band MSK modulation, which in Extended Mode is 250 kS/s and uses a nominal modulation index of 0.5. Transmission filtering may be required in order to meet the stopband requirements for a given Channel Class.

7.2.2.4 Normal MSK Modulation

Certain channel classes use a Normal-Band MSK modulation, which in Extended Mode is 100 kS/s and uses a nominal modulation index of 0.5. Transmission filtering may be required in order to meet the stopband requirements for a given Channel Class.

7.2.2.5 Narrow-Band MSK Modulation

Certain channel classes use a Narrow-Band MSK modulation, which in Extended Mode is 31.25 kS/s and uses a nominal modulation index of 0.5. Transmission filtering may be required in order to meet the stopband requirements for a given Channel Class.

7.2.2.6 Normal Channel Class

The Normal Channel Class provides for up to five concurrent channels, using MSK Modulation on Channel Center Frequency Indices 0x01, 0x04, 0x07, 0x0A and 0x0D. The Channel Bandwidth is always 324 kHz. CSMA-CA is optional, prior to transmission.

7.2.2.7 Wide-Band Channel Class

The Wide-Band Channel Class permits up to three concurrent channels, using Extended Mode high data rate MSK Modulation on Channel Center Frequency Indices 0x02, 0x07, 0x0C. The Channel Bandwidth is always 540 kHz. CSMA is optional, prior to transmission.

7.2.2.8 Narrow-Band Channel Class

The Narrow-Band Channel Class permits up to fifteen concurrent channels, using Extended Mode low data rate MSK Modulation. The Channel Bandwidth is always 108 kHz. CSMA is optional, prior to transmission.

7.2.3 CCA Process

Certain Channel Classes may make use of a CSMA routine prior to transmission of data over the channel, and for others it is optional. Upper layers of the specification, particularly the Data Link Layer and Transport Layers, describe the processes that utilize CSMA and Collision Avoidance models (CSMA-CA).

Table 162 — Extended Mode Conformance to IEEE 802.15.4-2011 Clear channel Assesment

Clause of IEEE 802.15.4f-2012	Title	Conformance	Comment
8.2.7.	Clear Channel Assesment	Yes	

7.2.4 PHY Layer Packet Structure

Extended Mode Packets contain a Preamble, Sync Word, Payload Length, Payload, and CRC. Additionally, power ramp-up and ramp-down periods may precede and follow the packet. These periods should be kept as short as possible, and used only for the purpose of meeting the channel stopband requirements presented in 7.2.2.2.2.

PHY Packet is constructed as specified in IEEE 802.15.4-2011 with specifics relevant to 433 MHz PHY which are specified in IEEE 802.15.4f-2012.

Table 163 — Extended Mode Conformance to IEEE 802.15.4-2011 PHY Layer Packet Structure

Clause of IEEE 802.15.4-2011	Title	Conformance	Comment
9.	PHY Services	-	
9.1.	Overview	Yes	
9.2.	PHY Constants	Yes	

Table 164 — Extended Mode Conformance to IEEE 802.15.4f-2012 PHY Layer Packet Structure

Clause of IEEE 802.15.4f-2012	Title	Conformance	Comment
16.1	PPDU Formats	Yes	
16.2	Data Rate	Yes	
16.3	SFD for MSK PHY	Yes	

PHY layer packet structure as specified by above tables is shown in Table 166.

Table 165 — PHY Packet Structure

Preamble	Sync Word	Payload Length	Payload	CRC
32 symbols	16 symbols	8 bits (1 byte)	0 – 96 bytes	16 bits (2 bytes)

7.2.4.1 Preamble

The Preamble is a series of 32 binary symbols, 10101…1010. used for the purpose of calibrating data rate circuits on the receiver and reliable detection of packet start. The Preamble always starts with binary 1 and ends with binary 0.

7.2.4.2 Sync Word

The Sync Word is a block of 16 symbols (which are well intended binary), chosen for excellent correlation properties when following the Preamble.It is used to align the frame.

7.2.4.3 Payload Length

Payload Length is an eight bit field consisting of a 7 bit value containing the length of the Payload field, and the eighth bit (MSB) reserved.

7.3 MAC Layer

This clause specifies the Extended Mode MAC sub-layer. It is based and fully compliant with IEEE 802.15.4e-2012 standard specification. Only some elements of IEEE 802.15.4e-2012 specification are utilized in Extended Mode as described in this clause. For full specification and implementation details refer to IEEE 802.15.4e-2012 standard specification.

These are the functions of the MAC Layer:

— Generating network beacons if the device is a gateway or subcontroller

— Synchronizing to network beacons

— Supporting network association and disassociation

— Supporting device security

— Employing appropriate mechanism for channel access

— Providing a reliable link between two peer MAC entities

— Providing management utility to access PHY layer and the layer above the MAC

7.3.1 Requirements of industrial and other application domains - IEEE 802.15.4e-2012 Features

Several behaviors are provided in IEEE 802.15.4e-2012 Draft for different industrial and other application domains and functional improvements compared to IEEE 802.15.4:2011.

The different industrial and other application domains have quite different requirements that are often in conflict with each other such that the resulting solutions cannot be the same. That is the rationale for specifying more than one solution because there is more than one problem to solve.

The intentions of these behaviors are to enhance and add functionality to the IEEE 802.15.4-2011 MAC to

- better support the industrial markets and
- permit compatibility with modifications being proposed within the Chinese WPAN.

Industrial applications have requirements that are not adequately addressed by the IEEE 802.15.4-2011 standard such as low latency, robustness in the harsh industrial RF environment, and determinism.

The Chinese Wireless Personal Area Network (CWPAN) standard has identified enhancements to improve network reliability and increase network throughput to support higher duty-cycle data communication applications.

Specifically, the MAC enhancements are grouped into two categories:

- Industrial and other application domains such as Process automation, Factory automation and
- Additional functional improvements such as Low energy.

The MAC enhancements are:

a) Time Slotted Channel Hopping (TSCH), e.g. for Process automation, see I.2 and I.6.

b) Low Latency Deterministic Networks (LL), e.g. for Factory automation, see I.3.

c) Distributed Synchronous Multi-Channel Extension (DSME), e.g. for Process automation and Commercial applications, see I.4.

d) RFID Blink (RFID), e.g. For item and people identification, location, and tracking, see I.7.

e) Asynchronous Multi-Channel Adaptation (AMCA), see I.8.

Additional functional improvements are:

- Low Energy (LE), optional, see I.5.
- Enhanced Beacon request (EBR), optional, see I.6.
- Enhanced Security and Overhead Reduction (ESOR)
- MAC Performance Metrics (Metrics)
- Fast association (FastA)
- Asynchronous Multi-Channel Adaptation (AMCA), see I.8

Annex I of IEEE 802.15.4e-2012 Draft provides material for a better understanding for the different solutions specified in this chapter.

7.3.2 MAC frame formats

The MAC frame format specified in this clause shall:

— support the clear separation of the higher protocol layers from the MAC layer,

— support encapsulation of any network layer protocol (e.g. IPv4, 6LowPAN/IPv6) in the MAC

— support direct encapsulation of the Application layer

— Allows various forms of addressing to be deployed

This subclause specifies the format of the MAC frame (MPDU). Each MAC frame consists of the following basic components:

a) A MHR, which comprises frame control, sequence number, address information, and security related information.

b) A MAC payload, of variable length, which contains information specific to the frame type. Acknowledgment frames do not contain a payload.

c) A MFR, which contains a FCS.

7.3.3 General MAC frame format

Extended Mode MAC frame structure is adopted from IEEE 802.15.4-2011 and addendum IEEE 802.15.4e-2012 standard. Description provided here is shown for informational purposes only. For full specification, refer to the IEEE 802.15.4-2011 and IEEE 801.15.4e standards.

Note: IEEE 802-15.4e-2012 standard is an addendum to previously published IEEE 802.15.4-2011 standard. It is mandatory to take into account all aspects of both standards when constructing Extended Mode MAC frames. For clarity and simplicity reasons, references given sometimes call only clauses in

IEEE 802.15.4e-2012. However, these clauses in many cases have their own calls to relevant clauses in IEEE 802.15.4-2011.

A complete MAC frame consists of the MAC Header, MAC Payload, and FCS as shown in Table 166. See 7.2 for a description on the PHY Header and CRC.

Table 166 — General MAC frame format

PHY Header	MAC Header	MAC Payload	FCS

7.3.3.1 MAC Header

Normative specification of MAC header is given in IEEE 802.15.4e-2012 standard. Table 167 lists relevant clauses.

Table 167 — Extended Mode Conformance to IEEE 802.15.4e-2012 MAC Header

Clause of IEEE 802.15.4e-2012	Title	Conformance	Comment
5.2.1.	General MAC Frame Format	Yes	All sub-clauses under clause 5.2.1. are relevant to Extended Mode MAC Frame format.

A MAC header consists of fields shown in Table 168. Unless specified different, all frame fields shall be present.

Table 168 — MAC Frame Header Format

Frame Control	Sequence Number	Dest. Network Identifier	Destination Address	Source Network Identifier	Source Address	Security Header	Information Elements

The Frame Control field contains information that specifies the type of MAC frame and the usage of other fields within the MAC frame header. The Frame Control field is described in clause 7.3.3.3.

The Sequence Number field is an incrementing value that is unique for each and every MAC frame.

A MAC frame header shall contain a minimum of zero and maximum of four of the address fields: Source Address, Destination Address, Source Network Identifier, and Destination Network Identifier. If all address fields in MAC Header are supressed, alternate address methodology shall be provided in either the Information Elements field or in Payload .

The Security Header shall be present if the Security bit is a value of 1 in the Frame Control field.

Information Elements field are used to encapsulate frame specific data. Devices can accept or discard a particular element if the element ID is unknown. Information Elements (IE) are divided into “Header IE” and “Payload IE”. Only “Header IE” belongs in MAC Header.

7.3.3.2 MAC Payload

The MAC Payload field contains the encapsulated data specific for the MAC frame type. Optionally, application data can be tunnelled through any of the network and transport layer protocols. Refer to 7.3.3.4 for more information on use of the network and transport layers.

Table 169 — MAC data frame using optional LLC, Network, and Transport layers

MAC Header	MAC Payload			
Refer to 7.2	Network Frame Control	Network Frame Type (optional)	Network Address-ing (optional)	Application Data (optional)

Network Frame Control defines what fields are included in MAC payload.

Network Frame Type defines what kind of higher layer addressing is used (LLC, network, transport) and if Application Data is present.

7.3.3.3 Frame Control

Frame type is specified by Frame Control Field. General format of Frame Control Field (FCF) is specified in clauses referenced in Table 170.

Table 170 — Extended Mode Conformance to IEEE 802.15.4e-2012 Frame Control Field (FCF)

Clause of IEEE 802.15.4e-2012	Title	Conformance	Comment
5.2.1.1.	Frame Control Field	Yes	All sub-clauses under clause 5.2.1.1. are relevant to Extended Mode Frame Control Field

7.3.3.3.1 Beacon, Data, Acnowledgement and Command Frames

All 4 frame types specified in IEEE 802.15.4-2011 are amended by IEEE 802.15.4e-2012 with addition of Information Elements. Table 171 provides references for constructing Beacon, Data, Acknowledgement and Command frame types.

Table 171 — Extended Mode Conformance to IEEE 802.15.4e-2012 Beacon, Data, Acknowledgement, Command Frames

Clause of IEEE 802.15.4e-2012	Title	Conformance	Comment
5.2.2.1.	Beacon Frame Format	Yes	
5.2.2.2.	Data Frame Format	Yes	
5.2.2.3.	Acknowledgement Frame Format	Yes	
5.2.2.4.	MAC Command Frame Format	Yes	

7.3.3.3.1.1 Beacon Frame

Beacon Frame is sent by Network Coordinator to synchronize network devices and advertise network capabilities.

A coordinator can transmit network beacons in a beacon-enabled PAN. The MAC payload contains the superframe specification; GTS fields, pending address fields, and beacon payload. The MAC payload is prefixed with a MAC header (MHR) and appended with a MAC footer (MFR). The MHR contains the MAC Frame Control field, beacon sequence number (BSN), addressing fields, and optionally the auxiliary security header. The MFR contains a 16-bit frame check sequence (FCS). The MHR, MAC payload, and MFR together form the MAC beacon frame (i.e., MPDU).

The MAC beacon frame is then passed to the PHY as the PHY service data unit (PSDU), which becomes the PHY payload. The PHY payload is prefixed with a synchronization header (SHR), containing the Preamble Sequence and Start-of-Frame Delimiter (SFD) fields, and a PHY header (PHR) containing the

length of the PHY payload in octets. The SHR, PHR, and PHY payload together form the PHY packet (i.e., PPDU).

Reference IEEE 802.15.4e-2012 Clause 5.2.2.1.

7.3.3.3.1.2 Data Frame

Data Frame is general purpose frame sent to exchange data between two devices.

The payload of a data frame shall contain the sequence of octets that the next higher layer has requested the

MAC sub-layer to transmit.

The data payload is passed to the MAC sublayer and is referred to as the MAC service data unit (MSDU).

The MAC payload is prefixed with an MHR and appended with an MFR. The MHR contains the Frame

Control field, data sequence number (DSN), addressing fields, and optionally the auxiliary security header.

The MFR is composed of a 16-bit FCS. The MHR, MAC payload, and MFR together form the MAC data frame, (i.e., MPDU).

The MPDU is passed to the PHY as the PSDU, which becomes the PHY payload. The PHY payload is prefixed with an SHR, containing the Preamble Sequence and SFD fields, and a PHR containing the length of the PHY payload in octets. The preamble sequence and the data SFD enable the receiver to achieve symbol synchronization. The SHR, PHR, and PHY payload together form the PHY packet, (i.e., PPDU).

Reference IEEE 802.15.4e-2012 Clause 5.2.2.2.

7.3.3.3.1.3 Command Frame

The MAC payload contains the Command Type field and the command payload. The MAC payload is prefixed with an MHR and appended with an MFR. The MHR contains the MAC Frame Control field, DSN, addressing fields, and optionally the auxiliary security header. The MFR contains a 16-bit FCS. The MHR, MAC payload, and MFR together form the MAC command frame, (i.e., MPDU).

The MPDU is then passed to the PHY as the PSDU, which becomes the PHY payload. The PHY payload is prefixed with an SHR, containing the Preamble Sequence and SFD fields, and a PHR containing the length of the PHY payload in octets. The preamble sequence enables the receiver to achieve symbol synchronization. The SHR, PHR, and PHY payload together form the PHY packet, (i.e., PPDU).

The following Command Frames are defined:

- Association Request is sent by un-associated device when it want to associate with the network
- Association Response is generated by a device which controls the network which can accept or reject the association request
- Disassociation Notification is sent by a device which wants to terminate its association with the network
- Data Request is sent by a device which wants to send unsolicited data to other network device.
- PAN ID Conflict Notification is sent by a device by a device to the network coordinator when a Network identifier conflict is detected.
- Orphan Notification command is used by an associated device that has lost synchronization with its coordinator.

- Beacon Request Command is used by a device to locate all coordinators within its radio communications range during an active scan.
- GTS Request Command is used by an associated device that is requesting the allocation of a new Guaranteed Time Sloty or the deallocation of an existing GTS from the PAN coordinator.

Reference IEEE 802.15.4e-2012 Clause 5.2.2.4.

7.3.3.3.1.4 Acknowledgement Frame

Acknowledgement Frame is sent as either in response to a received Data Request Command or in response to receiving Data Frame or Command Frame.

The MAC acknowledgment frame is constructed from an MHR and an MFR; it has no MAC payload. The MHR contains the MAC Frame Control field and DSN. The MFR is composed of a 16-bit FCS. The MHR and MFR together form the MAC acknowledgment frame (i.e., MPDU).

The MPDU is passed to the PHY as the PSDU, which becomes the PHY payload. The PHY payload is prefixed with the SHR, containing the Preamble Sequence and SFD fields, and the PHR containing the length of the PHY payload in octets. The SHR, PHR, and PHY payload together form the PHY packet, (i.e., PPDU).

The Sequence Number field shall contain the value of the sequence number received in the frame for which

the acknowledgment is to be sent.

Reference IEEE 802.15.4e-2012 Clause 5.2.2.3.

7.3.3.3.2 Low Latency Frame

Low Latency frames are used to minimize overhead in latency critical networks. Low latency frames have a 1 octet frame control and are designed for minimal overhead through the use of 1-octet destination addresses.

The Sub Frame Type field is 2 bits in length and indicates the type of the low latency frame. For deteailed description of Low Latency Frame type and usage refer to IEEE 802.15.4e-2012 specification.

Table 172 — Extended Mode Conformance to IEEE 802.15.4e-2012 Low Latency Frame

Clause of IEEE 802.15.4e-2012	Title	Conformance	Comment
5.2.2.5.	Low Latency Command Frame Format	Yes	All sub-clauses under clause 5.2.2.5. are relevant

7.3.3.3.3 Multipurpose Frame

Construction of Multipurpose Frame is specified in Table 173.

Table 173 — Extended Mode Conformance to IEEE 802.15.4e-2012 Multipurpose Frame

Clause of IEEE 802.15.4e-2012	Title	Conformance	Comment
5.2.2.6.	Multipurpose Frame Format	Yes	All sub-clauses under clause 5.2.2.5. are relevant

The Multipurpose frame is similar to the Data frame, but the frame control field allows for more options to control 4 addressing fields (including suppression of all 4 addressing fields and using application specific addressing in IE or Payload), suppression of sequence number, and inclusion of IEs.

Multipurpose Frame allows minimum packet length by utilizing single octet frame control (instead of 2-octet frame control field) and optional suppression of all other fields, including address fields. Multipurpose Frame is used to transmit various frame types.

Two implementations of Multipurpose Frame which are particularly important fopr Extended Mode are:

- Blink Frame
- RFID Wake-Up Frame

7.3.3.3.3.1 Blink Frame

The Blink is a periodic transmission from active RFID tags that is received by RFID interrogator infrastructure. Multipurpose Frame can be configured in such a way to serve as Blink Frame as specified by Table 174.

Table 174 — Extended Mode Conformance to IEEE 802.15.4e-2012 Blink Frame

Clause of IEEE 802.15.4e-2012	Title	Conformance	Comment
5.2.2.7.	Multipurpose Blink Frame	-	
5.2.2.7.1.	General	Yes	
5.2.2.7.2.	Blink Payload Field	Yes	

The Blink frame has been designed to be of minimal size to make its transmission require the lowest amount of energy to give maximum battery life to RFID tags.The MHR for a Blink frame shall typically only contain a 1-octet, Frame Control Field, the Sequence Number Field, and the Source Address field. In its shortest form, Blink Frame has format as shown in Table 175.

Table 175 — Multipurpose Blink Frame of minimal size

Octets: 1	1	8	0-127	2
Frame Control	Sequence Number	Source Address	Payload	FCS

The Blink frame provides a mechanism for a device to communicate its ID (i.e. the IEEE EUI-64 Source Address as defined) and/or an alternate ID (in payload field, as referenced in Table 174), and optionally additional payload data to other devices without prior association and without an acknowledgment. The frame can be used by “transmit only” devices to co-exist within a network, utilizing Aloha protocol. Any devices that are not interested in this Blink have an opportunity to reject the frame at early stage during frame processing and not burden the MAC or higher communication layers with this, potentially high volume, data traffic.

7.3.3.3.3.2 RFID Wake-Up Frame

The RFID Wake-up frame is used by a Network Coordinator to wake up a specific device or a group of devices (all devices with the same PAN ID). This frame is constructed as per rules for constructing LE-Multipurpose Wakeup Frame.

Table 176 — Extended Mode Conformance to IEEE 802.15.4e-2012 LE-Multipurpose Wake-up Frame

Clause of IEEE 802.15.4e-2012	Title	Conformance	Comment
5.2.2.8.	LE-Multipurpose Wake-Up Frame	-	
5.2.2.8.1.	General	Yes	
5.2.2.8.2.	Wake-up frame payload	Yes	

RFID Wake-Up Frame is using Information Elements field to specify the time when the next frame will be sent. Use of Information Elements is specified in Table 176

Table 177 — Extended Mode Conformance to IEEE 802.15.4e-2012 Information Elements

Clause of IEEE 802.15.4e-2012	Title	Conformance	Comment
5.2.4.	Information Element	--	
5.2.4.1.	General	Yes	Extended Mode is using only IE as part of MAC Header
5.2.4.2.	Header Information Elements	Yes	Element ID = 0x1D is used (Randez-vous Time)
5.2.4.10.	Randezvous Time IE	Yes	

The Rendezvous Time (RZ Time) header Information Element is 2 octet field. The RZ Time is the expected length of time in units of 10 symbols between the end of the transmission of the RFID Wakeup frame and the beginning of the transmission of the payload frame (if present). The RZ Time shall be set by the next higher layer when requesting the MAC sublayer to transmit. The last wakeup frame in a wakeup sequence shall have RZ Time set to the value zero. If no data is intended to follow RFID Wakeup frame, RZ Time shall be set to value zero.

7.3.3.4 Network and transport layer routing of application data

Application Data can be routed through any of the network and transport layer protocols. This is accomplished by setting Network Protocol type to appropriate protocol, including the LLC, network, and transport headers within the Mac Payload field, preceding the application data (see 7.3.3.2).

In this scenario the LLC protocol type will specify network layer protocol such as 6LowPAN, IPv4, etc. The network layer will then specify the transport layer protocol such as TCP or UDP. Finally, the transport layer protocol will specify the port number for the application. Note that inclusion of these protocol headers within the MAC frame may increase the size of the frame significantly, so this can be used if either the MAC frame size is big enough or if the IP (Internet Protocol) connectivity is mandatory. To reduce the MAC frame size, the 6LowPan can be used with compression. This is a standard method for encapsulation of network and transport layer protocols. These protocols (802.11 LLC, 6LowPAN, IP, UDP, TCP, etc) are used as defined by their standards, and there is no need for modification of these protocols for support in Extended Mode.

7.3.4 Channel Access

This subclause describes the mechanisms for channel access which includes scheduling, and contention based and contention free access of the channel. Contention-based access allows devices to access the channel in a distributed fashion using a CSMA-CA back-off algorithm.

7.3.4.1 Superframe structure

Use of superframe structure is optional for Extended Mode. The format of superframe is defined as per IEEE 802.15.4-2011. Relevant clauses are specified in Table 177.

Table 178 — Extended Mode Conformance to IEEE 802.15.4-2011 Superframe Structure

Clause of IEEE 802.15.4-2011	Title	Conformance	Comment
4.5.1.	Superframe structure	Yes	
4.5.2.	Data Transfer Model	Yes	
4.5.2.1.	Data Transfer to a Coordinator	Yes	
4.5.2.2.	Data Transfer from a Coordinator	Yes	
4.5.2.3.	Peer-to-Peer Data Transfer	Yes	

The format of the superframe is defined by the coordinator. The superframe is bounded by network beacons sent by the coordinator and is divided into equally sized slots. Optionally, the superframe can have an active and an inactive portion. The beacon frame is transmitted in the first slot of each superframe. If a coordinator does not wish to use a superframe structure, it will turn off the beacon transmissions. The beacons are used to synchronize the attached devices, to advertize the capabilities of the network, and to describe the structure of the superframes.

For low-latency applications or applications requiring specific data bandwidth, the coordinator may dedicate portions of the active superframe to that application. These portions are called guaranteed time slots (GTSs). The GTSs form the contention-free period (CFP), which always appears at the end of the active superframe starting at a slot boundary immediately following the contention access period (CAP).

For networks supporting beacons, synchronization is performed by receiving and decoding the beacon frames. For networks not supporting beacons, synchronization is performed by polling the coordinator for data.

Detailed description of how to use superframe structure is is specified in Table 178.

Table 179 — Extended Mode Conformance to IEEE 802.15.4-2011 Superframe Structure

Clause of IEEE 802.15.4-2011	Title	Conformance	Comment
5.1.1.	Channel Access	Yes	
5.1.1.1.	Superframe Structure	Yes	
5.1.1.1.1.	Contention Access Period (CAP)	Yes	
5.1.1.1.2.	Contrntion-Free Period (CFP)	Yes	
5.1.1.2.	Incoming and outgoing superframe timing	Yes	
5.1.1.3.	Interframe Spacing	Yes	

7.3.5 Data transfer model

As specified in Table 177, three types of data transfer transactions exist:

- Data Transfer to a coordinator
- Data transfer from a coordinator

- Peer-to-peer data transfer

7.3.5.1 Data transfer to a coordinator

When a device wishes to transfer data to a coordinator in a beacon-enabled PAN, it first listens for the network beacon. When the beacon is found, the device synchronizes to the superframe structure. At the appropriate time, the device transmits its data frame, using slotted CSMA-CA, to the coordinator. The coordinator may acknowledge the successful reception of the data by transmitting an optional acknowledgment frame. This sequence is summarized in Figure 22.

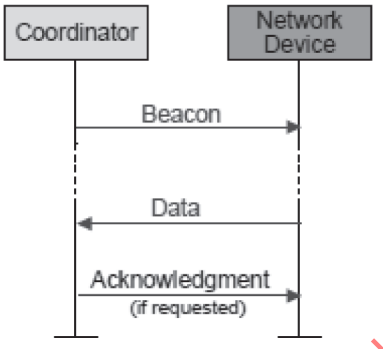


Figure 22 — Communication to a coordinator in a beacon-enabled PAN

When a device wishes to transfer data in a nonbeacon-enabled PAN, it simply transmits its data frame, using unslotted CSMA-CA, to the coordinator. The coordinator acknowledges the successful reception of the data by transmitting an optional acknowledgment frame. The transaction is now complete. This sequence is summarized in Figure 23.

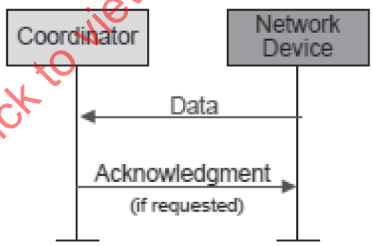


Figure 23 — Communication to a coordinator in a nonbeacon-enabled PAN

7.3.5.2 Data transfer from a coordinator

When the coordinator wishes to transfer data to a device in a beacon-enabled PAN, it indicates in the network beacon that the data message is pending. The device periodically listens to the network beacon and, if a message is pending, transmits a MAC command requesting the data, using slotted CSMA-CA. The coordinator acknowledges the successful reception of the data request by transmitting an acknowledgment frame. The pending data frame is then sent using slotted CSMA-CA or, if possible, immediately after the acknowledgment. The device may acknowledge the successful reception of the data by transmitting an optional acknowledgment frame. The transaction is now complete. Upon successful completion of the data transaction, the message is removed from the list of pending messages in the beacon.

This sequence is summarized in Figure 24.

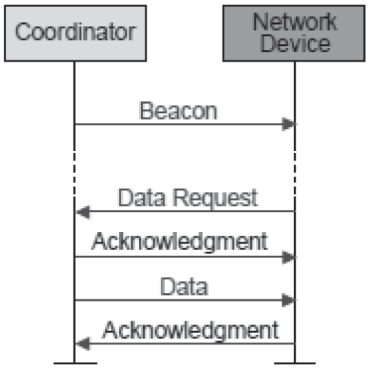


Figure 24 — Data transfer from a coordinator

When a coordinator wishes to transfer data to a device in a nonbeacon-enabled PAN, it stores the data for the appropriate device to make contact and request the data. A device may make contact by transmitting a MAC command requesting the data, using unslotted CSMA-CA, to its coordinator at an application-defined rate.

The coordinator acknowledges the successful reception of the data request by transmitting an acknowledgment frame. If a data frame is pending, the coordinator transmits the data frame, using unslotted CSMA-CA, to the device. If a data frame is not pending, the coordinator indicates this fact either in the acknowledgment frame following the data request or in a data frame with a zero-length payload. If requested, the device acknowledges the successful reception of the data frame by transmitting an acknowledgment frame. This sequence is summarized in Figure 25.

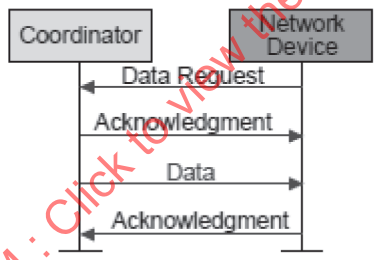


Figure 25 — Communication from a coordinator a beacon-enabled PAN

7.3.5.3 Peer-to-peer data transfers

In a peer-to-peer PAN, every device may communicate with every other device in its radio sphere of influence. In order to do this effectively, the devices wishing to communicate will need to either receive constantly or synchronize with each other. In the former case, the device can simply transmit its data using unslotted CSMA-CA. In the latter case, other measures need to be taken in order to achieve synchronization.

Such measures are beyond the scope of this standard.

7.3.5.4 CSMA-CA mechanism

Two types of CSMA-CA channel access mechanism can be used, depending on the network configuration: Beacon and Non-Beacon. Both of these mechanisms are using the methods specified in Table 180.

Table 180 — Extended Mode Conformance to IEEE 802.15.4-2011 CSMA-CA Mechanism

Clause of IEEE 802.15.4-2011	Title	Conformance	Comment
4.5.4.1.	CSMA-CA Mechanism	Yes	
5.1.1.4.	CSMA-CA Algorithm	Yes	

7.3.5.5 ALOHA mechanism

Aloha mechanism allows for transmission of unacknowledged and unassociated frames. This method of network access is particularly important for very low power RFID devices.

7.3.5.6 Frame acknowledgment

A successful reception and validation of a data or MAC command frame can be optionally confirmed with an acknowledgment. If the receiving device is unable to handle the received data frame for any reason, the message is not acknowledged.

If the originator does not receive an acknowledgment after some period, it assumes that the transmission was unsuccessful and retries the frame transmission. If an acknowledgment is still not received after several retries, the originator can choose either to terminate the transaction or to try again. When the acknowledgment is not required, the originator assumes the transmission was successful.

Detailed specification of frame acknowledgemtn and retransmission is provided in Table 181

Table 181 — Extended Mode Conformance to IEEE 802.15.4-2011 Frame Acknowledgement and Retransmission

Clause of IEEE 802.15.4-2011	Title	Conformance	Comment
5.1.6.4.	Use of Acknowledgement and Retransmission	Yes	
5.1.6.4.1.	No Acknowledgement	Yes	
5.1.6.4.2.	Acknowledgement	Yes	
5.1.6.4.3.	Retransmission	Yes	

7.3.5.7 Data verification

In order to detect bit errors, an FCS mechanism employing a 16-bit International Telecommunication

Union—Telecommunication Standardization Sector (ITU-T) cyclic redundancy check (CRC) is used to detect errors in every frame as per reference in Table 182.

Table 182 — Extended Mode Conformance to IEEE 802.15.4-2011 Data Verification

Clause of IEEE 802.15.4-2011	Title	Conformance	Comment
5.2.1.9.	FCS Field	Yes	

7.3.6 MAC Security

MAC level security for Extended Mode is specified by references in Table 183.

Table 183 — Extended Mode Conformance to IEEE 802.15.4-2011 MAC Security

Clause of IEEE 802.15.4-2011	Title	Conformance	Comment
4.5.6.	Security	Yes	
7	Security	Yes	

7.3.6.1 Frame Security for Extended Mode

Besides the frame security support defined in the IEEE 802.15.4:2011 this standard provides additional frame security methods and algorithms as defined in ISO/IEC 29167. The Frame Security for Extended ISO 18000-7 does not interact with the Frame Security defined in the IEEE 802.15.4. Each method may be used without any interaction with each other.

This security protocol consists of following components:

- Mutual Authentication with key derivation procedure
- Frame Security

Mutual Authentication with key derivation procedure is a part of the Application Framework. All Mutual authentication commands are encapsulated in MAC Data or Mutlipurpose frames. All processing and key derivation process is performed at the application layer. The application part of this protocol is described in section 7.4.7 of this document.

Frame Security is performed at the MAC Layer providing confidentiality and integrity of wireless frames. The MAC payload (including application layer data) is encrypted and authenticated.

7.3.6.1.1 Extended Mode Security Information Element

In order to extend MAC functionality to support the Extended ISO 18000-7 Security Services, Extended Mode Security Information Element is defined and its presence in the MAC header is marked by setting *IE List Present bit* in the Frame Control field of the MAC header. The Security Information Element is processed at the MAC Layer.

IEs can be managed and unmanaged. For the purpose of specifying Extended Security services, Extended Mode is using Unmanaged IE with Element ID = 0x01.

The construction of IE field is determined by references in Table 184.

Table 184 — Extended Mode Conformance to IEEE 802.15.4-2011 Information Elements

Clause of IEEE 802.15.4-2011	Title	Conformance	Comment
5.2.4.	Information Element	-	
5.2.4.1.	General	Yes	
5.2.4.2.	Header Information Elements	Partial	Unmanaged IE = 0x01 is used for Extended Security Services.
5.2.4.3.	Payload Information Element	No	Only Header IE is used.

Extended ISO 18000-7 Security IE Content is shown in Table 185 — Extended ISO 18000-7 Security IE Content

Table 185 — Extended ISO 18000-7 Security IE Content

1 byte	16/8 bytes
Security Options	IV/CCM Header

- Security Options field is defined in Security Options section.
- IV/CCM Header: The initialization vector used to encrypt the frame with a symmetric algorithm.

7.3.6.1.2 Security Options

Security Options field is defined in Table 186.

Table 186 — Security Options

Bits: 0	1	2	3	4	5	6	7
Reserved	Reserved	Secure Extended ISO 0= Frame is not encrypted (there is no IV and authenti-cation field in the frame) 1= Frame is encrypted	Protection suites *See* Table 187			Reserved	PKI *See* 7.3.6.1.4 0=PKI not used 1=PKI used

The Secure Extended ISO bit, described in the above table, shall be used to indicate if the frame is secured or not. If the value of the bit 3 is:

- If the Secure Extended Mode bit is set to 1 then the frame is secured and it contains Extended Mode Security IE with Security Options and IV/CCM fields, as well as Encrypted Payload, and Authentication Data present in the frame. The Application command code and command arguments are encrypted within the Encrypted Payload field.
- If the Secure Extended Mode bit is set to 0 then the frame is NOT secured and Extended ISO 18000-7 Security IE, Encrypted Payload, and Authentication Data are NOT present in the frame. Application data containing command code and command arguments are NOT encrypted.

7.3.6.1.3 Extended Mode protection suites – Symmetric cryptography

Cryptographic protection of the Extended Mode messages consists in an optional message encryption step followed by a message authentication step. Each protection suite type is assigned an enumerated descriptor also called a protection suite identifier. A protection suite identifier is written as a designator of an encryption method followed by a designator of an authentication method:

<message-encryption-method>-<message-authentication-method>

The protection suite descriptor uniquely identifies the methods used to authenticate and encrypt the message.

Table 187 contains some of the currently considered crypto protection suites.

Table 187 — Extended Mode Protection Suites

Protection Suite Descriptor	Encryption Algorithm	Key Size, bit	Encryption Mode	Authentication Algorithm	Authentication size, # bits	Protection suites bits
AES-128-CBC-SHA1-96	AES	128	CBC	SHA1-96	96	001

Table 187 *(continued)*

Protection Suite Descriptor	Encryption Algorithm	Key Size, bit	Encryption Mode	Authentication Algorithm	Authentication size, # bits	Protection suites bits
AES-128-CBC-SHA1	AES	128	CBC	SHA1	160	010
CCM -7 (CCM for ISO/ IEC 18000-7)	AES	128	Counter	AES-CBC-MAC	64	011
HB2-128	HB2-128	128	Normal	HB2-128	16-128	100
NULL – NULL (secure bit is 0)	None	N/A	N/A	None	N/A	000

Packet Options (Protection suites bits) values 100, 101, 111 are for future extensions.

CCM-7 is based on CCM, a generic authenticated encryption block cipher mode of AES. CCM is a mode of operation defined for any block cipher with a 128-bit block size. CCM combines two well-known and proven cryptographic techniques to achieve robust security. First, CCM uses CTR for confidentiality and Cipher Block Chaining MAC (CBC-MAC) for both authentication and integrity protection.

7.3.6.1.4 Extended Mode protection suites – Asymmetric cryptography

When PKI is used, PKI bit in the Packet Options is set to 1. Protection suite bits will code the PKI algorithm used for mutual authentication:

001 ECC

010 RSA

Packet Options (Protection suites bits) values 011, 100, 101, 111 are for future extensions.

7.3.6.1.5 Encryption process

Once a device is mutually authenticated and the session keys are generated as described in 7.4.7.1 Mutual Authentication section, the device can exchange MAC data and multupurpose frames with encrypted and authenticated paylaod.

The Application layer sends a packet as defined in Table 201 with Application Header and Payload to the MAC layer. MAC layer checks the device configuration. If e.g. the device is configured to use AES for encryption and SHA1 for authentication, MAC will generate an Initialization Vector (IV) and create an Extended ISO 18000-7 Security Information Element (see 7.3.6.1.1). The Extended ISO 18000-7 Security Information element will contain the IV and Security Options - *Protection suites* field initialized with value AES-128-CBC-SHA1. The Information Element Header will be initialized with the *the Security Element Id and IE Content Length. IE List Present bit* in the Frame Control field of the MAC header will be set to one. Then, MAC will generate authentication field caculating SHA1 over the the packet received from the Application Layer, and MAC will encrypt the packet using AES algorithm with the session keys generated during Mutual Authentication Process.

Encrypted and authenticated payload of the Multipurpose/Data frame is shown in Table 188 — Encrypted and Authenticated MAC Mutlipurpose/Data Frame Payload. The size of the Authentication field depends on the selected authentication algorithms.

Table 188 — Encrypted and Authenticated MAC Mutlipurpose/Data Frame Payload

N byte	12/20/8 bytes
Encrypted MAC Payload	Authentication Data (SHA1-96|SHA1|CBC-MAC)

- **Encrypted MAC Payload:** The encrypted application layer header and payload.

- **Authentication Data:** The hash function result calculated over the payload prior to encryption.

7.3.6.1.6 Decryption process

If MAC receives a Multipurpouse/ Data frame with both ***IE List Present bit*** in the Frame Control field of the MAC header set to one and with the ***the Security Element*** present in the MAC header***.*** MAC will decrypt the payload using IV provided in ***the Security Element*** and algorithms in Security Options - ***Protection suites.*** Then MAC will re-calculate authentication field over decrypted payload and compare it with one received in the frame. If the authentication check is successful, then decrypted payload will be sent to the Application Layer. If the authentication check is not successful, the frame will be descarded.

7.3.7 Wake on Mechanisms

Following Wake on mechanisms may be supported:

- UHF Wake on
- LF Wake on
- Sensor/Alarm wake on
- Additional wake on

7.3.7.1 UHF Wake on

As defined by the ISO 18000-7, the Wake Up Signal is transmitted by the interrogator for a minimum of 2.4 seconds to wake up all tags within communication range. The Wake Up Signal consists of a 2.3 to 4.8-second 31.25 kHz square wave modulated signal called the "***Wake Up Header***" immediately followed by a 0.1-second 10 kHz square wave modulated signal called the "***Co-Header***." Upon detection and by completion of the Wake Up Signal all tags enter into the Ready state awaiting a command from the interrogator. A tag has two states, awake/ready and asleep. During the ready state, the tags accept valid commands from interrogators and respond accordingly. When the tag is asleep, it ignores all commands.

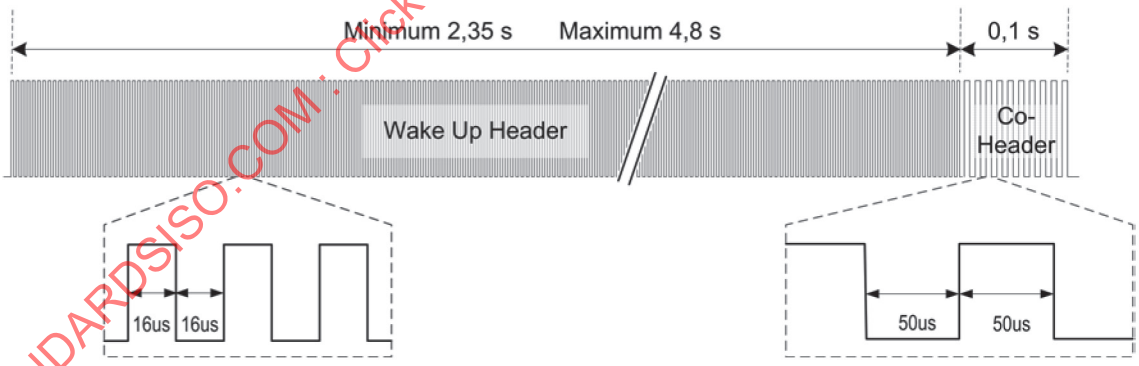


Figure 26 — Wake Up Signal

Once awoken, the tag shall stay awake for a minimum of 30 seconds after receipt of the last well-formed message packet consisting of a valid Extended protocol ID, command code, and CRC values, unless the interrogator otherwise commands the tag to sleep. If no well-formed command message is received within the 30 seconds, the tag will transition to the sleep state and SHALL no longer respond to command messages from Interrogators.

The communication between interrogator and tag shall be of the Master-Slave type, where the interrogator shall initiate communications and then listen for a response from a tag.

7.3.7.1.1 Distributed Sliding Wake-up Algorithm

Instead of sending complete 2.4 sec wakeup signal, an interrogator can distribute the wakeup signal into multiple smaller wakeup units, and distribute sending of these Distributed Wakeup Signals following the Distrubuted Wake-up Algorithm.

There two variants of the Distributed Wakeup Signal:

- One dimensional distribution of the wakeup signal
- Two dimensional distribution of the wakeup signal

7.3.7.1.2 One dimensional distribution of the wakeup signal

In ISO 18000-7, each RFID tag wakes up every 2.4 seconds and checks for a wakeup signal. If there is no wakeup signal, the tag will go back to sleep for additional 2.4 seconds. This process repeats until the tag receives the wakeup signal and wakes up. In the context of this algorthm, 2.4 second interval is called „wakeup super frame“

The 2.4 second interval, can be sub-divided into multiple beacon intervals, e.g. 24 intervals with 100 millisecond duration each. In a beacon enabled RF network, the interrogator shall send a beacon frame periodically every beacon interval. The interrogator will distrubute wakeup signal and embed the wake-up procedure into normal operating cycle.

One of the 100 millisecond beacon intervals from total of 24 in each wakeup super frame, will be dedicated to 100 millisecond wakeup signal. The wake-up signal is distributed between multiple wakeup super frames. In the first (index 0) wakeup super frame, the wake up signal will occupy beacon interval 0. In the second (index 1) wakeup super frame, the wake up signal will occupy the beacon interval 1. In the third (index 2) wakeup super frame, the wake up signal will take the beacon interval 2. Since the wake-up, a 100 millisecond interval slides, in the last (index 23) wakeup super frame, the wake up signal will occupy beacon interval 23. After each of the 100 millisecond wake-up signals, the RFID interrogator can perform the collection procedure of the woken up tag group before sending the next wake up signal.

It takes 24 wake-up intervals (100msec each) to wake up all tags.

In 57.6 sec all tags will be woken up and possibly collected. This would guarantee that newly arrived tags will be woken up (discovered) in 57.6 seconds (24 X 2.4sec = 57.6 sec).

Overhead for sending distributed-sliding wake-up signals is 1/24 equals 4.17%.

Figure 27 depicts the Distributed Sliding Wake-up algorithm for 100 millisecond beacon interval.

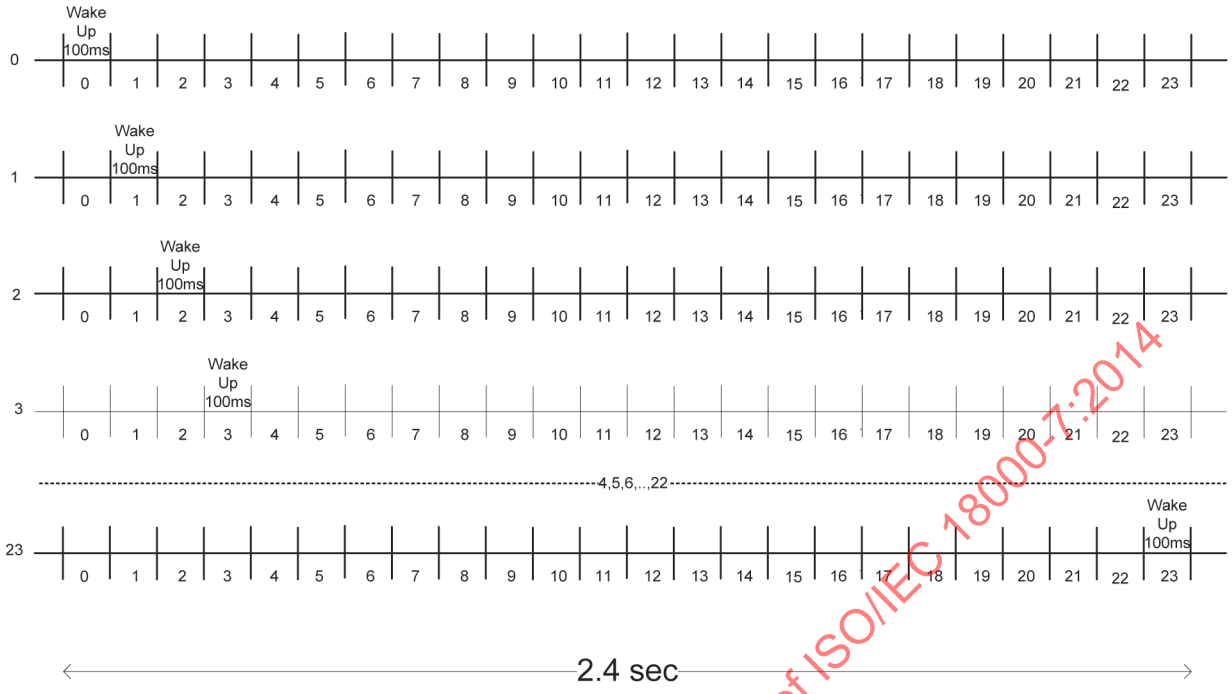


Figure 27 — Distributed Sliding Wake-up Algorithm

We assume that the wakeup interval slides in the frames 4 to 22 from slot 4 to slot 22.

The interrogator can take multiple of 2.4 second intervals for collection of a group of woken up tags between two sliding wake-ups. It is important that next 100msec wake up interval slides, so the next group of tags can be woken up.

Note: if the network is configured to be in beconless mode of operation, the intorrogator will not be sending beacons, and the distribution of the wakeup signal will remain the same as described.

7.3.7.1.3 Two dimensional distribution of the wakeup signal

In beacon enabled networks the wakeup signal can be reduced bellow a beacon interval. In Figure 28 the wake up signal is 1/6 of the beacon interval. In each wakeup super frame, the wakeup signals slide inside each beacon interval. The wakeup signal is distributed in two dimensions, both in a beacon interval and in a wakeup super frame, Figure 28 The RFID devices are divided into 24 groups, although the super frame is divided into 4 beacon intervals. In one dimensional distribution of the wakeup signal, there would be only four groups of devices. Two dimensional distribution shall even further better address the issue of evenly waking up a population of a large number of tags.

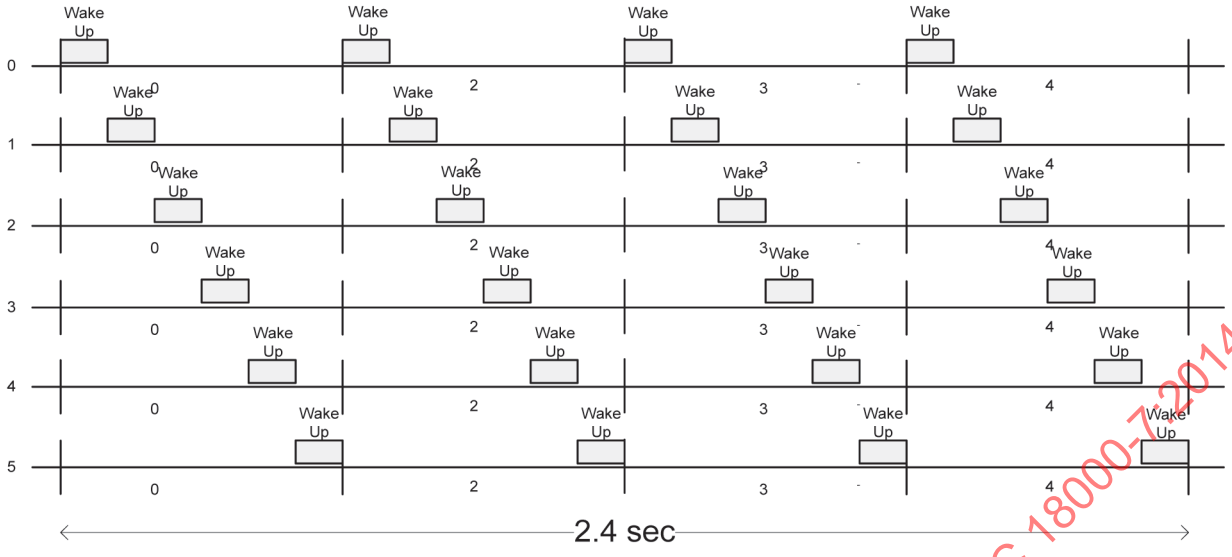


Figure 28 — Two dimensional distribution of the wakeup signal

Note: if the network is configured to be in the beconless mode of operation, the intorrogator will not be sending beacons, and the distribution of the wakeup signal will remain the same as described.

7.3.7.1.4 Distributed Wake-up Signal in Extended ISO 18000-7

Existing ISO 18000-7 defined Wakeup signal with scaled down Wake-Up Header and Co-Header duration can be used re-used for Distributed Sliding Wakeup Algorithm as presented in the Table 189 — Distributed Wakeup Signals.

Table 189 — Distributed Wakeup Signals

Wake Up Type	Wake Up Header square wave modulated signal		Co-Header square wave modulated signal	
	Duration	Frequency	Duration	Frequency
ISO 18000-7 Wakeup signal	2.3 to 4.8-second	31.25 kHz	0.1-second	10 kHz
Distributed Wake Up signal	T_{FW} = Configurable <=1.2 sec	F_{FW} 31.25 kHz	T_{FC} <= 0.1 second	F_{FC} = 10 kHz

7.3.7.2 LF Wake-On

The Low Frequency (LF) link supports short range communication from an exciter to the tag. This LF link enables the system to wake-up the tag and issues commands to the tag. As LF is a one-way link, the tag's response to the LF command is transmitted through the UHF link making use of the Extended Mode PHY, MAC and Application framework.

7.3.7.2.1 Exciter to Tag Communication (LF link)

The exciter to tag communication utilizes low frequency (122.64 kHz) AM modulation scheme and operates at short range. Data is transmitted from the exciter to the tag without acknowledgment over LF (one way LF communication link).

7.3.7.2.1.1 Data modulation and coding

Data transmitted between the exciter and the tag utilizes OOK scheme with two distinctive signal levels (RF carrier frequency being switched on or off).

7.3.8 Preamble

The preamble shall be comprised of eight (8) cycles each 16T period long (where T = 1/32768 sec). The preamble shall start with signal high and then alternating between 8T high and 8T low periods.

The preamble shall be preceded by RF Burst (RF ON) of minimum duration of 48T followed by 8T of signal low (RF OFF). Refer to Figure 2

Note that when multiple packets are being transmitted back to back the RF Burst signal is required to be transmitted only for the first transmitted packet.

7.3.9 Data bytes

Data bytes shall be in Manchester code format, each byte is comprised of 8 data bits. The data bit period shall be 16T (where T = 1/32768 sec), the total byte period shall be 128T. A falling edge in the centre of the bit-time indicates a 0 bit, a rising edge indicates a 1 bit.

7.3.10 Packet end period

A final period of 24T of continuous high, followed by a 24T continuous low and followed by minimum of 8T high shall be transmitted after the last Manchester encoded bit within the packet.

=

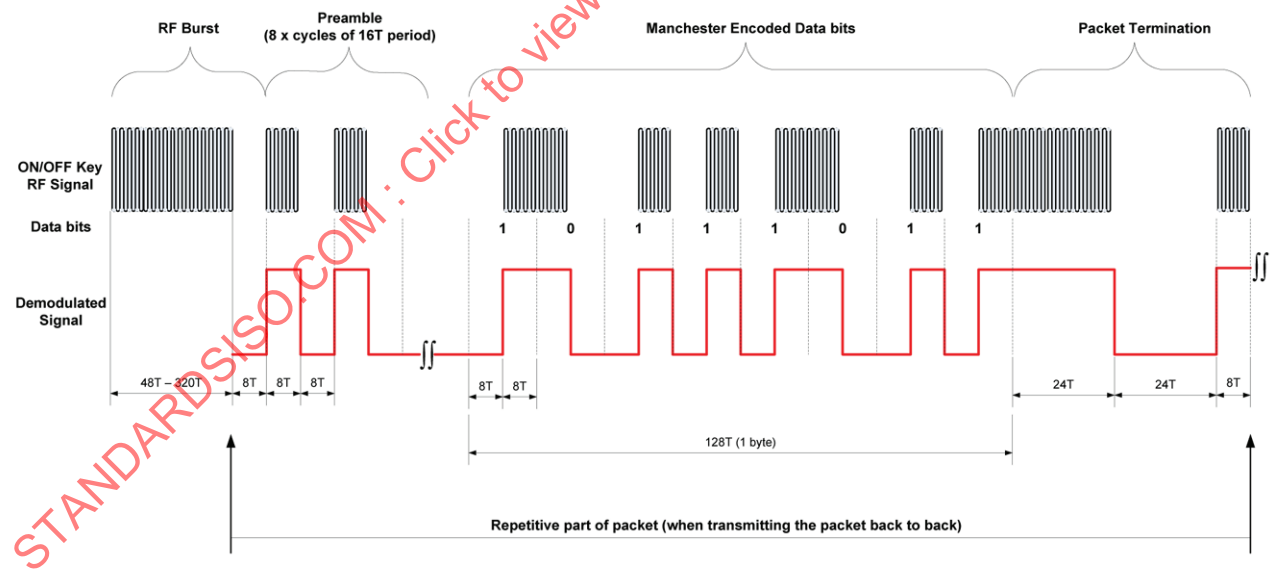


Figure 29 — Data Modulation

The bit period shall have duration of 16T, where T = 1/32768 seconds with an error tolerance of +/-3%. This symbol rate of produces effective data rate of 2.048 kb/s.

The packet structure is as shown in Figure 31.

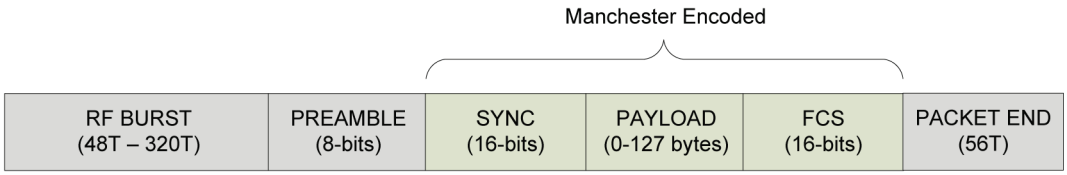


Figure 30 — Packet Structure

Notes:

The order of transmitted Data bytes is in accordance with 802.15.4e MAC specification.

The packet transmission starts with transmission of carrier signal (CW) in duration of 48T periods to 320T periods. Acctual duration of CW transmission is application dependant and can be used to trade off RFID tag response time versus its power consumption.

Peramble field (P) follows CW signal and is 8 bits long (all “0” when Manchester encoded).

Sync Word field serves to detect start of frame and shall be 0x115F (binary value = 0001 0001 0101 1111) . The Preamble of 0x00 and Sync Word of 0x115F work together to provide accurate start of packet (start of frame delimitation).

A CRC checksum shall be calculated as a 16-bit value for each packet, initialized with all zeroes (0x0000), over entire payload bytes (excluding preamble and sync word) according to the CCITT polynomial (x16 + x12 + x5 +1). The CRC shall be appended to the data included in the command message as a two bytes field. Reference: ITU-T Recommendation V.41 (Extract from the Blue Book), Code-independent error-control system, Appendix I - Encoding and decoding realization for cyclic code system.

7.3.10.1.1.1 Protocol

The type and amount of data being transmitted from the exciter to the tag is dependent on the particular application. Several modes of operation are being defined to provide flexible message structure to meet various application requirements.

7.3.10.1.1.2 Message structure

The exciter to tag communication protocol uses byte oriented, packet based message structure utilizing 16-bit Frame Check Sequence (FCS) error detection mechanism for reliable communication. The protocol utilizes 16-bit mode field prefix that defines the message structure and optimizes the packet size that is being sent to the tag.

The message structure with all the fields is shown in Table 190.

Table 190 — Payload message structure

LF Mode	Application ID (Optional)	Tag ID (Optional)	Exciter ID	LF RSSI (Optional)	Exciter Time-out (Optional)	Command Code (Optional)	Parameters (Optional)	
8-bits/16-bits	8-bits	48-bits / or 64-bit IEEE ID	16-bits	8-bits	8-bits	8-bits	Variable	16-bits

Notes:

The transmission of the packet fields within the message starts with the LF Mode field and ends with the FCS field. The most significant bit within each byte is sent first. Lower order byte is sent first within each field. The FCS calculation includes all bytes in the packet except itself.

7.3.10.1.1.3 LF Mode field

The LF mode field bits are defined Table 191

Table 191 — LF Mode field (byte 1)

Value	'0'	'1'	Definition
bit 0	No Application ID	Application ID	This bit indicates whether the Application ID is included or excluded from the message.
bit 1	No Tag ID	Tag ID	This bit indicates whether the Tag ID field is included or excluded from the message.
bit 2	48-bit Tag ID	64-bit IEEE ID	This bit indicates whether 48-bit Tag ID or 64-bit IEEE address is being used..
bit 3	LF RSSI threshold not used	LF RSSI threshold used	This bit indicates whether LF RSSI threshold value (1 byte) is included in the message. The RSSI threshold level informs a tag on boundary of the LF field at which it shall respond (tag's measured LF RSSI is greater than LF RSSI threshold). If the measured RSSI is lower than provided LF RSSI threshold a tag will not respond. If this field is not included in the message a tag shall respond to all correctly received LF messages regardless of the signal level.
bit 4	No Command	Command	This bit indicates whether the Command and associated parameters fields are included in the message.
bit5 & bit 6	LF Sequence ID		LF Sequence ID is used to identify all identical messages sent by the exciter as a part of the same transaction (same Sequence ID).
bit 7	Extension Mode byte not included	Extension Mode byte included	This extension bit is used to expand LF Mode byte with additional 8-bit of bitmapped link control bits. If 2nd LF Mode byte is not included than it is assumed that bit options defined in 2nd LF Mode byte are set to zero.

The LF mode field bits are defined Table 191

Table 192 — LF Mode field (byte 2)

Value	'0'	'1'	Definition
bit 0	Exciter Timeout not used	Exciter Timeout is used	Exciter Timeout field is included in the packet.
bit 1	Exciter IN	Exciter IN and OUT	This option instructs the tag to generate one or two types of UHF responses depending on the operational scenarios. If Exciter IN and OUT response is selected than tag will send two UHF messages, IN when entering the Exciter field and OUT when leaving the Exciter field. If Exciter IN response is selected than the tag will generate only IN UHF message as it passes through the Exciter.
bit 2	Reserved	Reserved	
bit 3	Reserved	Reserved	
bit 4	Reserved	Reserved	
bit 5	Reserved	Reserved	

Table 192 *(continued)*

Value	'0'	'1'	Definition
bit 6	Reserved	Reserved	
bit 7	Reserved	Reserved	

7.3.10.1.1.4 Message Parameters

- Application ID indicates the tag command set to be used. If this field is not included in the message, the Application ID is defaulted to 0x02.
- Tag ID is the 48 bit or 64-bit ID of the target tag, depending if Base Mode addressing or IEEE EUI-64 addressing is used
- .
- Exciter ID is a 16-bit number that uniquely identifies the exciter. The range of the Exciter IDs is from 1 to 2^{16}-1. The Exciter ID zero is reserved.
- Exciter Timeout represents the exciter OFF time during its periodic transmission. It is used to prevent the tag to retransmit the same message for every exciter transmission cycle. The range of the Exciter Timeout is from 1 to 255 seconds.
- Command Code follows a simple structure with only two main command categories:
- Write to the tag
- Read from the tag

Where command format is defined as follows:

Table 193 — Command format

Command prefix (1-bit)	Tag Command code (7-bits)
1 – Write 0 – Read	*See Section 7.3.10.1.1.5.*

Parameters is a variable length field that contains the parameter(s) associated with the LF Command. The length of the parameter field is derived based on the command code and is not explicitly specified.

7.3.10.1.1.5 Tag Command Code

The exciter can issue tag commands from multiple tag command sets. The Application ID indicates the tag command set to be used.

Table 194 describes the default tag command set that is associated with Application ID 0x02. Note that if no Application ID is included in the LF packet, this default tag command set should be assumed. The definition of additional tag command sets is outside of the scope of this specification.

Default tag command set:

Table 194 — Default tag command set for Application ID 0x02

Tag Command Codes	Tag Parameter Name	Tag Parameter Size [bits]	Parameter Type Description	R/W Status
0x07	User ID byte length	8	User ID byte length indicates the length of the ISO User ID data field. When the User ID Length is set to 0 the User ID is disabled and will not be reported within the UHF response message.	R
0x0B	Tag type	8	Tag Type is used to distinguish various tag models and is manufacturer specific.	R
0x0E	Wake Up	-	This command is only applicable for tag that supports 2-way UHF communication. Upon reception of this command tag will turn on UHF receiver and listen for incoming UHF command from the Coordinator.	-
0x31 / 0xB1	Tag transmission period under the Exciter field	1+15	The most significant bit is used to enable or disable tag periodic exciter IN transmissions while tag is still under Exciter LF filed. The lowest 15 bits indicate period in seconds. If all 15 bits are zero than no change in transmission rate is assumed and only most significant bit is used to enable or disable transmission rate.	R/W
0x32 / 0xB2	Tag beacon transmission period	1+15	The most significant bit is used to enable or disable periodic tag beacon transmission. The lowest 15 bits indicate period in seconds. If all 15 bits are zero than no change in transmission period is assumed and only most significant bit is used to enable or disable transmission rate.	R/W
0x43 / 0xC3	Manufacturer specific.			
0x98	Manufacturer specific.			
0x99	Manufacturer specific.			
0x9A	Manufacturer specific.			
0x2A / 0xAA	Manufacturer specific.			

7.3.10.1.2 Tag Response (UHF link)

The tag's response to the LF packet is encapsulated within an Application Data Packet and is carried by the MAC layer using MAC Data frame or MAC Multipurpose frame.

The data being sent to the coordinator has a different format depending on the mode of operation. The LF response application payload format is shown in Table 195.

Table 195 — LF response appliation payload format

Application ID	Tag Status	Tag ID	Exciter ID	LF RSSI (Optional)	User ID (Optional)	Command Code (Optional)	Command Arguments (Optional)
8-bits	16-bits	48-bits / 64-bit IEEE ID	16-bits	8-bits	16-bits	8-bits	Variable

Transmission order is least significant byte (LSB) first (within multi-byte field) while within a byte the order is least significant bit first.

7.3.10.1.2.1 Message parameters

- Application ID is equal to the Application ID in the LF packet. If no Application ID is received, the default value of 0x02 should be included in this field.
- Tag Status (see descriptions in Section 7.3.10.1.2.2)
- Tag ID is the 48-bit or 64-bit ID of the transmitting tag, depending if Base Mode addressing or IEEE EUI-64 addressing is used.
- Exciter ID is the 16-bit Exciter ID in the LF packet received from the exciter.
- User ID is an alternate ID used to address the tag and is manufacturer specific.
- Command Code is the same command code in the LF packet received from the exciter.
- Command Arguments is the response to the LF command. Its length depends on the particular LF command.

7.3.10.1.2.2 Tag Status

The Tag Status word is transmitted in every tag UHF response. It represents a collection of various bit fields indicating the current tag operational states and packet format as defined in in Table 196.

Table 196 — Tag status word

Bit							
15	14	13	12	11	10	9	8
UHF Mode field				Reserved=0	Reserved=0	IEEE Addressing=0 Base Mode Addressing=1	Ack Flag 1 = Nack 0 = Ack
Bit							
7	6	5	4	3	2	1	0
UHF Sequence ID		Tag type			Tag beacon mode 1 = enabled 0 = disabled	User ID 1 = yes 0 = no	Service bit

- Service bit when set (‘1’) indicates that the tag has detected a hardware-related fault condition.
- Tag beacon mode indicates whether tag’s beacon mode is enabled or disabled.
- User ID flag indicates whether a manufacturer specific User ID is used to identify a tag.
- Tag type is used to distinguish various tag models and is manufacturer specific.
- UHF Sequence ID is used to identify all identical messages sent by the tag as a part of the same transaction.
- Acknowledgment flag is used to notify the system that a valid command has been received from the Exciter.
- UHF Mode Field (see below)

7.3.10.1.2.3 UHF Mode Field

The data being sent to the system has a different format depending on the mode of operation.

The UHF mode field indicates the tag's mode of operation. Table 197 specifies the available UHF mode:

Table 197 — Available UHF modes

UHF Mode field format code (bit3, bit2, bit1, bit0)	UHF Mode
0000	Exciter IN
0001	Tag Beacon
0010	Command
0011	Exciter OUT
01xx	Manufacturer Specific
1xxx	Reserved

7.3.10.1.2.4 Tag response message format

Two distinctive tag response formats are used depending on the LF Mode field bit 2 setting (No Command or Command).

7.3.10.1.2.5 Exciter no-command tag response format

If the exciter packet does not contain command field, then the UHF response has the following structure:

Table 198 — Exciter no-command tag response format

Application ID	Tag Status	Tag ID	Exciter ID	User ID*
8-bits	16-bits	48-bits/64-bits	16-bits	N x 8 bits

** Optional field depending on whether User ID is enabled within the tag.*

7.3.10.1.2.6 Exciter command tag response format

If the exciter packet contains command field, then the UHF response has following structure:

Table 199 — Exciter command tag response format

Application ID	Tag Status	Tag ID	Exciter ID	Command Code	Command Parameters*
8-bits	16-bits	48-bits/64-bits	16-bits	8-bits	N x 8 bits

** Optional field depending on the command code*

The command code is the same command code received from the exciter. The command parameters field is optional and its length depends on the particular command issued by the exciter.

7.4 Application layer Framework

This clause specifies the Extended Mode Application layer which provides a framework and services for user applications to employ. It is responsible for the following tasks:

— Defines resident resources and provides core services to access and manage resources

— Routing of application data to/from MAC to resident applications

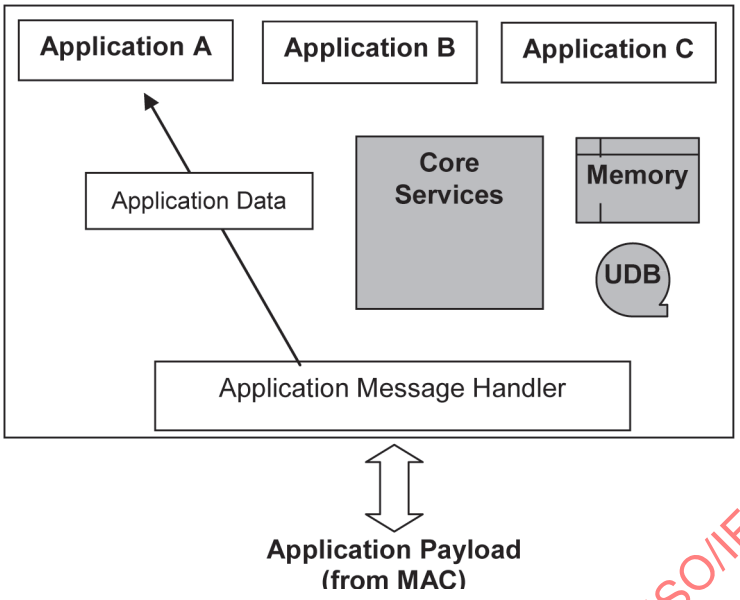


Figure 31 — Application Layer Framework

7.4.1 General Application data packet format

The Application Data Packet consists of a MAC or Mutlipurpose Data Frame with a MAC Header and MAC Payload. An Application Data Packet consisting of an Application Header and Application Payload is embedded within the MAC Payload along with optional LLC.

Table 200 — MAC Data Packet format with Application Data

MAC Header					
MAC Data Frame type or MAC Multipurpose Frame type (refer to 7.3.3.1)	LLC (optional)	6LowPAN or IPv4 Header (optional)	UDP/TCP Header (optional)		
				Application Header	Application Payload

An Application Header consists of fields shown in Table 201. Unless specified, all frame fields shall be present.

Table 201 — Application Data Format

Extended protocol ID	Options/ Tag Status	Alternate Addressing (optional)	Payload Data
1 byte	1 / 2 bytes	Variable	0-N

The Application header consists of Extended protocol ID, Options (in request packets) or Tag Status (in response packets) and Payload Length field.

7.4.2 Extended protocol ID

Extended protocol ID presents a unique identifier for a specific application. Following extended protocol IDs are carried over from ISO 18000-7 as application IDs.

Table 202 — Extended protocol ID

Extended protocol ID	Standards
0x01	18000-7 version 1
0x02	LF
0x03	Security Services
0x04	Alternate Addressing

In order to support any of above application a separation of the application layer from the MAC and PHY has to be done. In this standard, the Application Layer relies on IEEE 802.15.4 MAC functionality. The network is created on the MAC layer. Once the network is created, the application layer on infrastructure and end devices can exchange application layer commands to perform required functions.

7.4.3 Creating a wireless network

The IEEE 802.15.4 standard defines the methods for creating Beacon enabled and Beaconless wireless networks. This process may include association procedure of wireless devices as well as synchronization procedure. Depending of a specific use case, wireless devices create a network with a coordinator and then exchange application layer messages. Application layer messages are carried in the payload of MAC Data frames. Application layer will configure the MAC and PHY depending on the application use case.

7.4.3.1 Tag Initiated Communications

Association process can introduce additional overhead especially if devices are quickly moving and spending very little time in a network. For such cases, there may not be sufficient time to associate to the network and exchange data. So, there is a need for having an optional association procedure. One method supported by IEEE 802.15.4 which allows for the exchange of frames without an prior association procedure is the use of the multipurpose frame. Multipurpose frames are treated in the same manner as Data frames, and their content is passed to/from the next higher layer using the MCPS-DATA primitives. This procedure is called the ***Tag Initiated Communication***. A Tag can initiate communication as a result of any internal event or alarm and it can use multipurpose or blink frames for sending data to the interrogator without being previously associated to the network.

Moreover, a tag may initiate a communication as a result of a wake on mechanism such as those described in clause 7.3.7.

7.4.4 ISO 18000-7 Application Support

In order to support ISO 18000-7 functionality, the application layer has to be separated from the MAC and PHY as described in this section of the document. The commands without dependencies on the MAC and the PHY remain unchanged. However, some commands and procedures dependent on the MAC/PHY functionality are modified.

The fields of the original ISO 18000-7 packets are mapped as follows:

- ***ISO 18000-7 Extended protocol ID*** *is mapped into the* Extended protocol ID field of the Application header
- ***ISO 18000-7 Command Code and Command Arguments*** are mapped into the application Payload

Application header and payload fields for legacy ISO 18000-7 type of application are defined in Table 203.

Table 203 — Application payload format for legacy ISO 18000-7 type of application

Application Header	Application Payload				
Extended protocol ID	Options/ Tag Status	Tag Manufacturer ID (optional)	Tag Serial Number (optional)	Command Code	Command Arguments
0x01	1 / 2 bytes	2 bytes	4 bytes	1 byte	N bytes

7.4.4.1 Extended protocol ID

Extended protocol ID is mapped into the Extended protocol ID field of the Application header.

The extended protocol ID field 0x01 for support of legacy ISO/IEC 18000-7 applications.

7.4.4.2 Command Codes

The Command codes and their function as a Read and/or Write command shall be as listed in Table 4, below. Codes not identified are reserved.

Table 204 — Command codes

Command code + Sub Command Code (R / W)	Command name	Command type	Mandatory/Optional		Description
			Interrogator	Tag	
0x1F / NA	Collection with Universal Data Block	Broadcast	Mandatory	Mandatory	Collects all Tag IDs and Universal Data Block
NA / 0x15	Sleep	Point to Point	Mandatory	Mandatory	Puts tag to sleep
NA / 0x16	Sleep All But	Broadcast	Mandatory	Mandatory	Puts all tags but one to sleep
0x13 / 0x93	User ID	Point to Point	Mandatory	Optional	Sets user assigned ID (1 – 60 bytes)
0x09 / 0x89	Routing Code	Point to point	Mandatory	Mandatory	Reads and writes routing code
0x0C / NA	Firmware Version	Point to Point	Mandatory	Optional	Retrieves manufacturer-defined tag firmware revision number
0x0E / NA	Model Number	Point to Point	Mandatory	Optional	Retrieves manufacturer-defined tag model number
0x60 / 0xE0	Read/Write Memory	Point to Point	Mandatory	Optional	Reads and writes user memory
NA / 0x95	Set Password	Point to Point	Mandatory	Optional	Sets tag password (4 bytes long)
NA / 0x97	Set Password Protect Mode	Point to Point	Mandatory	Optional	Engages/disengages password protection (see section 6.3.4)
NA/ 0x96	Unlock	Point to Point	Mandatory	Optional	Unlocks password protected tag
0x70 / NA	Read Universal Data Block	Point to Point	Mandatory	Mandatory	Reads the Universal Data Block
0x26+0x01	Table Create	Point to Point	Mandatory	Optional	Creates a database table
0x26+0x02	Table Add Records	Point to Point	Mandatory	Optional	Prepares to add new records to the specified database table

Table 204 *(continued)*

Command code + Sub Command Code (R / W)	Command name	Command type	Mandatory/Optional		Description
			Interrogator	Tag	
0x26+0x03	Table Update Records	Point to Point	Mandatory	Optional	Prepares to modify the specified table records
0x26+0x04	Table Update Fields	Point to Point	Mandatory	Optional	Prepares to update the specified fields of a table record
0x26+0x05	Table Delete Record	Point to Point	Mandatory	Optional	Deletes existing record from the existing database table
0x26+0x06	Table Get Data	Point to Point	Mandatory	Optional	Prepares to retrieve the specified table records
0x26+0x07	Table Get Properties	Point to Point	Mandatory	Optional	Gets total number of records and the maximum number of records the table can hold
0x26+0x08	Table Read Fragment	Point to Point	Mandatory	Optional	Retrieves a block of data from a table as initiated by the Table Get Data command
0x26+0x09	Table Write Fragment	Point to Point	Mandatory	Optional	Writes a block of data into a table as initiated by the Table Add Records, Table Update Records, or Table Update fields command
0x26+0x10	Table Query	Broadcast or Point to Point	Mandatory	Optional	Initiates table search based on the specified criteria
0xE1 / NA	Beep ON/OFF	Point to Point	Mandatory	Optional	Turns tag's beeper ON or OFF
0x8E	Delete Writeable Data	Point to Point	Mandatory	Optional	Deletes all allocated writeable data on a tag

The Command Type column indicates whether the command is broadcast or point-to-point. Once the payload is passed down the protocol stack, MAC layer will initialize addresses and frame control bits accordingly.

For commands requiring a Sub Command Code, the Sub Command Code field is the first byte of the Command Arguments field that follows the Command Code

7.4.4.3 Command Arguments

Some commands require arguments. For those commands where arguments are defined, argument data shall be supplied with the command. The contents and length of any required arguments are specific to each command.

7.4.4.3.1 Tag-to-interrogator payload format

The tag-to-interrogator message shall use one of two formats depending on the type of message being transmitted to the Interrogator. The tag shall always respond to a command using one of the response formats described below except in the following situations, for which the tag shall not respond:

— the command is explicitly specified as requiring no response

— receipt of a broadcast command containing an invalid command code or other error

— the tag is in the asleep state

There are two possible response formats:

— the Broadcast response message format

— the Point-to-Point response message format

7.4.4.3.1.1 Broadcast response application payload format

The application payload format shown in Table 205 shall be used in response to Interrogator broadcast commands received by tags within the Interrogator's communication range. Broadcast commands are identified in Table 203.

Table 205 — Broadcast response application payload format

Extended protocol ID	Tag Status	Command Code	Data
0x01	2 bytes	1 byte	N bytes

— **Tag Status:** Indicates various conditions such as response format, tag type, alarm and hardware fault. See section 7.4.4.3.2.1, Tag Status, for more details.

— **Command Code:** Command code (see Table 4) received from the Interrogator

— **Data:** Data returned by the tag as a response to an Interrogator's valid broadcast command request. The value of N, the length of the data in bytes, is specific to the command. In the event that the tag receives an invalid command, no response is sent to the interrogator

7.4.4.3.1.2 Point-to-point response application payload format

This application payload format, shown in Table 206, shall be returned to the Interrogator as a response to all point-to-point commands, which require the Tag Manufacturer and Serial Number in order to access a particular tag. (Point-to-point commands are identified in Table 203).

Table 206 — Tag–to-interrogator response application payload format (point-to-point)

Extended protocol ID	Tag Status	Command Code	Response Data*
0x01	2 bytes	1 byte	N bytes

*This field is command dependent; some commands may or may not need this field

— **Tag Status:** Indicates various conditions such as response format, tag type, alarm and hardware fault.

— **Command Code:** Command code received from the Interrogator

— **Response Data:** Data returned by the tag as a response to an Interrogator's valid command request. The value of N, the length of the data in bytes, is specific to the command. In the event an error is detected, a NACK flag within the Tag Status word will be set and the Response Data will contain an error response as described in Error codes subsection.

7.4.4.3.1.3 Error codes

In response to a point-to-point command a tag may reply with one of the errors listed in Table 207. If multiple errors are detected in a point-to-point command, only the first error is reported. Errors resulting from broadcast commands do not generate responses.

Table 207 — Error code

Error Code	Description
0x01	Invalid Command Code
0x02	Invalid Command Parameter
0x03	Optional Command not Supported
0x04	Not Found
0x06	Can't Create Object
0x08	Authorization Failure
0x09	Object is Read-Only
0x0A	Operation Failed
0x3f	Implementation Dependent
0x40	Stale Token
0x41	Boundary Exceeded

Error response data shall consist of a one-byte error code; possibly a one-byte sub-code, depending on the kind of error; possibly one or more bytes of parameter data, also depending on the error; and an optional, manufacturer-defined number of additional data bytes, as shown in Table 208. In the following error definition sections, the optional, manufacturer-defined data bytes are not shown.

Table 208 — General error format

Error Code	Sub-code	Error Parameter Data	Manufacturer Data
1 byte	1 byte	N bytes	M bytes

— **Error Code:** a value from Table 7 identifying the kind of error

— **Sub-code:** an optional value that further refines the nature of the error and is specific to the kind of error. This field is absent if the error does not define a Sub-code. Sub-codes are specified in the error description subsections below.

— **Error Parameter Data:** N bytes of data, where N is zero or greater, whose existence, length, and content depend on the nature of the error. This field is absent if the error does not define Error Parameter Data. Error specific Error Parameter Data and length N of this field, if any, is specified in the error description subsections below.

— **Manufacturer Data:** M bytes of data, where M is zero or greater, whose existence, field length, and content are at the discretion of the tag manufacturer

Invalid command code error

Table 209 shows the structure of this error code.

Table 209 — Invalid command code error

Error Code
0x01

This error as defined in Table 9 shall be generated when the tag receives a packet with a Command Code and/or Sub Command Code that is not defined in this part of ISO/IEC 18000.

Invalid command parameter error

Table 210 shows the structure of this error code.

Table 210 — Invalid command parameter error

Error Code	Sub-code	Parameter Offset
0x02	1 byte	1 byte

— **Sub-code:** a code as shown in Table 211 that describes the error more specifically. Following values are defined:

Table 211 — Invalid command parameter error sub-codes

Sub-code	Sub-error Name	Meaning
0x01	Parameter Out of Range	The value of a parameter is not legal
0x02	Too Few Parameters	There are fewer bytes in the Command Arguments field than expected
0x03	Too Many Parameters	There are more bytes in the Command Arguments field than expected

Parameter offset: the offset in bytes from the beginning of the Command Arguments field where the error was detected.

This error as defined in Table 10 shall be generated when the tag receives a command with invalid or malformed parameters. If more than one parameter is in error, the first invalid parameter shall be reported.

Optional Command Not Supported

Table 212 shows the structure of this error code.

Table 212 — Optional Command Not Supported error

Error Code
0x03

This error shall be generated when the tag receives an ISO optional command that is not supported on this tag.

Not found error

Table 213 shows the structure of this error code.

Table 213 — Not found error

Error Code	Sub-code
0x04	1 byte

Sub-code: a code as shown in Table 214 that describes the error more specifically. Following values are defined:

Table 214 — Not found error sub-codes

Sub-code	Sub-error Name	Meaning
0x01	Table Does Not Exist	There is no existing table for the table ID given
0x02	Record Does Not Exist	There are fewer records than the record number given
0x03	Field Does Not Exist	There are fewer fields than the field number given

7.4.4.3.2 Cannot create object error

Table 215 shows the structure of this error code.

Table 215 — Can't create object error

Error Code	Sub-code
0x06	1 byte

— **Sub-code:** a code as shown in Table 216 that describes the error more specifically. The following values are defined:

Table 216 — Can't create object error sub-codes

Sub-code	Sub-error Name	Meaning
0x02	Table Already Exists	The requested table ID is already in use
0x03	Out of Memory	There is insufficient memory in the tag to create the requested table
0x04	Table ID Reserved	The table ID provided is reserved, and not available for assignment to a table.

This error as shown in Table 216 shall be generated upon an unsuccessful attempt to create a database table.

Authorization failure error

Table 217 shows the structure of this error code.

Table 217 — Authorization failure error

Error Code
0x08

This error as shown in Table 217 shall be generated upon an invalid attempt to access a tag feature protected by a password.

Object is read-only error

Table 218 shows the structure of this error code.

Table 218 — Object is read-only error

Error Code
0x09

This error as shown in Table 218 shall be generated upon an attempt to modify some tag data entity for which the tag does not allow modifying operations.

Operation Failed error

Table 219 shows the structure of this error code.

Table 219 — Operation Failed error

Error Code	Sub-code
0x0A	1 byte

— **Sub-code:** a code as shown in Table 220 that describes the error more specifically. The following values are defined:

Table 220 — Operation Failed error sub-codes

Sub-code	Sub-error Name	Meaning
0x01	Write Failure	The Memory write operation failed.
0x02	Erase Failure	The Memory erase operation failed.
0x03	Memory Consistency	Memory corruption has been detected
0x04	Other Failure	Operation failed for other reason

This error as shown in Table 19 shall be generated upon the failure of a valid command to complete properly. This error shall only be reported if the command failed to complete and no other error has been reported.

Implementation dependent error

Table 221 shows the structure of this error code.

Table 221 — Implementation dependent error

Error Code	Sub-code
0x3F	1 byte

This error code as shown in Table 21 shall be reserved for tag manufacturers and applications to define for tag behaviour errors not covered by this part of ISO/IEC 18000 Extended Mode. At a minimum, the tag implementation shall include a Sub-code field. Sub-code and any additional fields of the error are left to the tag manufacturer and applications to specify.

Stale Token error

Table 22 shows the structure of this error code.

Table 222 — Stale Token error

Error Code
0x40

This error as shown in Table 22 shall be generated by the tag when a submitted Request Token is invalid due to an intervening modification that was made to the table for which the Request Token applies. These modifications include invocations of the following commands: Table Add Records, Table Update Records, Table Update Fields, and Table Delete Record.

Boundary exceeded error

Table 23 shows the structure of this error code.

Table 223 — Boundary exceeded error

Error Code	Sub-code
0x41	1 byte

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

— **Sub-code:** a code as shown in Table 24 that describes the error more specifically. The following values are defined:

Table 224 — Boundary exceeded error sub-codes

Sub-code	Sub-error Name	Meaning
0x01	Table Full	The table has been filled to the create-time allotment
0x02	Record Does Not Exist	The record has not been added yet
0x03	Fragment Overrun	The write operation completed with still more to write
0x04	Field Does Not Exist	The field does not exist

This error as shown in Table 23 and sub-code shown in Table 24 shall be generated upon an attempt to access a record outside of a valid boundary.

7.4.4.3.2.1 Tag status

The Tag Status field shown in Table 25, included in all tag–to-interrogator messages, shall consist of the following information:

Table 225 — Tag status field format

Bit							
15	14	13	12	11	10	9	8
Mode field				Alarm	Reserved	Base Mode addresing	Acknowledge-ment1 = NACK 0 = ACK
Bit							
7	6	5	4	3	2	1	0
Reserved		Tag type			Reserved	Reserved	Service bit

Note reserved fields are set to a value of "0"

— **Mode field** indicates the format (response to Broadcast command or response to Point-to-Point command) of the response data from the tag. The list of possible values is shown in Table 26.

Table 226 — Tag status field format

Mode field	Mode format code (bit 15 - 12)
Broadcast Command	0000
Point to Point Command	0010

— **Alarm** is intended as a general status bit indicating a non-command related reportable condition. If set ('1'), an alarm condition has been detected by the tag. The interpretation, actions to retrieve data and clear the alarm bit is defined by the tag vendor.

— **Base Mode Addressing**, when set, specifies that Base Mode addressing (manufacturer ID (2 bytes) and Tag ID (4 bytes) is used in application Application Header. When set, IEEE EUI-64 addressing in Multipurpose Frame should be turned off.

— **Acknowledgment,** when clear ('0'), indicates that the tag has received a valid command (CRC ok and all fields valid) from the Interrogator and processed the command successfully. If set ('1'), the command was invalid or the tag encountered an error during the processing of the command. Note that as described in section 6.2.6, the tag issues no response in the case of a CRC error.

— **Tag type** is a value assigned by, and meaningful only to, the tag manufacturer. The manufacturer can use this value to indicate manufacturer-defined special features.

— **Service** bit when set ('1') indicates that the tag has detected a hardware-related fault condition. Additional information on the hardware fault condition may be retrieved with the Hardware Fault Status UDB element.

7.4.4.4 Tag commands

7.4.4.4.1 Collection with Universal Data Block (UDB)

The Collection with Universal Data Block command shall be used to collect Tag Manufacturer ID and Tag Serial Numbers with the contents of a specified UDB data block. The format of the Collection with Universal Data Block (Collection with UDB) command shall be as shown in Table 227,

Table 227 — Collection with Universal Data Block command

Command Code	Windows size	Max Packet Length	UDB Type Code
0x1F	2 bytes	1 byte	1 byte

— **Window Size:** the number of 57,3 ms intervals to use for listening for tag responses in the collection algorithm. Encoded as an unsigned 16-bit integer, with a valid range of 1 to 512. ***This is a Base Mode specific parameter and not used in the Extended Mode mode of operation.***

— **Max Packet Length:** an integer in the range 20 to 255 inclusive that specifies the maximum value that a tag can use as the Packet Length field of it's response. Tags may select a different reply packet length as long as the length does not exceed the value of Max Packet Length. This parameter may be used to tune performance or to limit RF transmission times for compliance with regional RF regulatory requirements. The value 20, the size of a minimum tag response packet (the length 20 include 15 bytes for response packet overhead, 1 byte for the UDB Type Code value, 2 bytes for the Total UDB Length value and 2 bytes for the Requested Offset value), indicates no bytes of the UDB should be included in the tag response.

— **UDB Type Code:** identifies the requested UDB type. See Table 40 for a list of defined UDB types.

The tag shall select a random time slot based upon the Window Size and Max Packet Length values received. The tag shall respond with the Collect with Universal Data Block broadcast response message as shown in Table 28.

When this command is received the tag shall save all requested UDB data and ensure no change to the UDB data until all data has been sent.

Table 228 — Collection with Universal Data Block response

Command Code	UDB Type Code	Total UDB Length	Requested Offset	UDB data
0x1F	1 byte	2 bytes	2 bytes	N bytes

— **UDB Type Code**: identifies the requested UDB type.

— **Total UDB Length**: the total length, in bytes, of UDB data on the tag for the selected UDB Type.

— **Requested Offset**: the tag shall reply with the value zero for its response to a Collection with UDB command. All Collection with UDB commands will begin at the implied offset of zero and the tag shall respond with data beginning at the first byte of the requested UDB block and confirm this offset value with the value 0 for the Requested Offset field.

— **Universal Data Block data**: an initial portion of the Universal Data Block.

7.4.4.4.1.1 Universal Data Block

The Universal Data Block contains zero, one or more data elements which are referred to as TLD (Type, Length, Data) Elements, and are formatted as shown in Table 229. Each TLD element is identified with

a UDB Element Type ID (see Table 30). Non-present or zero length data elements shall not be included in the Universal Data Block. For example, if the length of the User ID is zero, no part of the User ID TLD shall be included in the UDB.

Table 229 — TLD element format

UDB Element Type ID	Length	Data
1 byte	1 byte	N bytes
UDB Element Type ID (see Table 30)	N (length of Data in bytes)	Data bytes

— **UDB Element Type ID:** identifies Data element, UDB Element Type IDs are defined in Table 30.

— **Length:** number of bytes in length of Data element.

— **Data:** the informational content of the TLD, such as a Routing Code or User ID.

The values for the UDB Element Type ID shall be as shown in Table 230

Table 230 — - UDB Element Type ID values

UDB Element Type ID (1 byte)	Description	Note
0x00 - 0x0A	Reserved	
0x10	Routing Code	The routing code as specified within this document
0x11	User ID	User ID as specified within this document
0x12	Optional Command List	A list of command codes for optional commands supported on this tag
0x13	Memory Size	Total and available memory on this tag
0x14	Table Query Size	The total number of Table Query elements supported on this tag
0x15	Table Query Results	Results for the previously executed Table Query
0x16	Hardware Fault Status	Hardware reset count, Watchdog reset count and Hardware Fault bitmap (including low battery flag) to provide additional information when the “service” bit is set in the tag Status word
0x17 - 0x7F	Reserved	These elements are reserved for future tag data elements
0x80 - 0xFE	Future extension	Reserved for future use
0xFF	Application Element	Application extensions

The Routing Code UDB Element (0x10) shall be as shown in Table 231.

Table 231 — Routing Code UDB Element

UDB Element Type ID	Length	Data
1 byte	1 byte	N bytes
0x10	N	Routing code data

The User ID UDB Element (0x11) shall be as shown in Table 232.

Table 232 — User ID UDB Element

UDB Element Type ID	Length	Data
1 byte	1 byte	N bytes
0x11	N	User ID data

The Optional Command List Element (0x12) shall be as shown in Table 233. The data returned in this TLD element is a list of one-byte command code values for the optional commands that are implemented on this tag.

Table 233 — Optional Command List Element

UDB Element Type ID	Length	Data
1 byte	1 byte	N bytes
0x12	N	N 1-byte command code values

The Memory Size Element (0x13) shall be as shown in Table 234. The data returned in this TLD is composed of three 4-byte values: the total number of bytes available for Read/Write Memory commands, the total number of bytes allocated for Table database memory, and the number of bytes currently available in the Table database memory (available memory size does not include overhead and simply reports the number of unused memory bytes).

Table 234 — Memory Size Element

UDB Element Type ID	Length	Data		
1 byte	1 byte	12 bytes		
0x13	0x0C	4 bytes	4 bytes	4 bytes
		R/W Memory	Total Table Memory	Available Table Memory

The Table Query Size Element (0x14) shall be as shown in Table 235. The 8-bit unsigned integer value returned in this TLD element represents the number of Table Query elements supported on this tag.

Table 235 — Table Query Size Element

UDB Element Type ID	Length	Data
1 byte	1 byte	1 byte
0x14	0x01	number of Table Query elements supported

The Table Query Results Element (0x15) shall be as shown in Table 236. The data returned in this TLD is available after the successful execution of a Table Query and includes a Query Status value, the Table ID for the queried table, the number of records matched in that table, and the index of the first matching record.

Table 236 — Table Query Results Element

UDB Element Type ID	Length	Data			
1 byte	1 byte	7 bytes			
0x15	0x07	1 byte	2 bytes	2 bytes	2 bytes
		Query Status	Table ID	Number of Records Matched	Index of First Matched Record

The values of the Query Status field shall be as shown in Table 237.

Table 237 — Query Status values

Query Status Value	Description
0x00	The Table Query operation was successful.
0x01	The tag did not execute the query because the tag did not receive a complete sequence of Table Query packets, or a command has been received by the tag that has invalidated any previous query results.
0x02	The tag received a complete sequence of Table Query packets but the tag cannot comply and did not execute the query (e.g. the Table ID is invalid on the tag or a Sequence ID value greater than the maximum number supported by the tag).
0x03	Partial Query Results. The Table Query operation started but has not completed. The Number of Records matched and Index of First Matched Record field represent partial results of the Query.
0x04–0xFF	Reserved.

The Hardware Fault Status Element (0x16) shall be as shown in Table 38. The data returned in this TLD is composed of three 1-byte values: the lifetime count of hardware resets, the lifetime count of Watchdog (firmware) resets, and the Hardware Fault bitmap. The Hardware Fault Bitmap is defined as shown in Table 238.

Table 238 — Hardware Fault Status Element

UDB Element Type ID	Length	Data		
1 byte	1 byte	3 bytes		
0x16	0x03	1 byte	1 byte	1 byte
		lifetime count of hardware resets	lifetime count of firmware resets	Hardware Fault Bitmap

Table 239 — Hardware Fault Bitmap

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
reserved	reserved	reserved	reserved	reserved	reserved	Memory Corruption Detected	Low Battery Detected

where:

— **Low Battery Detected (bit 0):** when set (‘1’) indicates that the tag battery is “low.” The exact meaning of “low” is implementation defined.

— **Memory Corruption Detected (bit 1):** when set (‘1’) indicates that the tag has detected a memory hardware fault condition.

— **Reserved (bits 2-7):** reserved for future use.

A UDB Type is a predefined collection of UDB Element Types. The Collection with UDB and Read UDB commands include a UDB Type argument that allows an application to select one of the available predefined collections of UDB data. All UDB Types may include additional Application Extension TLD Elements following the required TLD Elements. The values of the UDB Type shall be as shown in Table 240.

Table 240 — UDB Types

UDB Type	Description	UDB Elements included for this UDB Type
0x00	Transit data	Routing Code UDB element (Element Type 0x10), User ID UDB element (Element Type 0x11) and any Application defined UDB elements.

Table 240 *(continued)*

UDB Type	Description	UDB Elements included for this UDB Type
0x01	Capability data	Optional Command element (Element Type 0x12), Memory Size element (Element Type 0x13) and Table Query Size element (Element Type 0x14) and any Application defined UDB elements.
0x02	Query results	Table Query Results element (Element Type 0x15) and any Application defined UDB elements.
0x03	Hardware Fault data	Hardware Fault Status element (Element 0x16) and any Application defined UDB elements.

The Universal Data Block may optionally include one or more UDB Application Extension Blocks each encapsulating one or more TLDs, which are uniquely identified by the included Application ID (see Table 241). Any individual tag may support the extensions defined by multiple vendors (with appropriate licensing if required).

Table 241 — UDB Application Extension Block format

Application Extension Type ID	Application Extension Length	Extended protocol ID TLD Element	Application TLD Elements
1 byte	1 byte	N bytes	M bytes
0xFF	N + M bytes	TLD containing the Extended protocol ID Type and Extended protocol ID value	one or more Application defined TLDs

Where:

— **Application Extension Type ID:** The Application Extension Type ID defined in Table 230. This Application Extension ID identifies that all TLDs included within this UDB Application Block are identified by the included Extended protocol ID.

— **Application Extension Length:** The full length of UDB Application Extension Block in bytes, including the Extended protocol ID TLD, and the combined lengths of the included Application TLD elements.

— **Extended protocol ID TLD Element:** The Extended protocol ID TLD Element must be formatted as described in Table 29 and consists of an Extended protocol ID Type, a one-byte length field and a data field containing the Extended protocol ID value for the entity responsible for defining the following Application defined TLD elements. Extended protocol ID Types are defined as in Table 242.

— **TLD Elements:** A series of one or more TLDs each consisting of a Type ID byte defined by the included Extended protocol ID, a one-byte length field and a data field. The TLD Type IDs are defined solely by the Application identified, and are not required to be made public. All of the included TLDs must be formatted as described in Table 229, except that the Type ID is assigned by the manufacturer rather than this part of ISO/IEC 18000 Extended Mode. All of the included TLDs must fit completely within the Application Element Length byte count.

Table 242 — Extended protocol ID TLD Types

Extended protocol ID TLD Type code	Extended protocol ID TLD value
0x00	Manufacturer ID - the Extended protocol ID is the 16-bit Tag Manufacturer ID assigned by the Registration Authority as called out in ISO/IEC 15963.

Table 242 *(continued)*

Extended protocol ID TLD Type code	Extended protocol ID TLD value
0x01	Routing Code - The Extended protocol ID is the Tag Data Routing Code as defined in the ISO 17363 standards.
0x02 - 0xFF	Reserved

See Figure 3 for an example Universal Data Block of UDB Type 0x00. The example includes a Routing Code element, a User ID element and an Application extension block with two application extension elements.

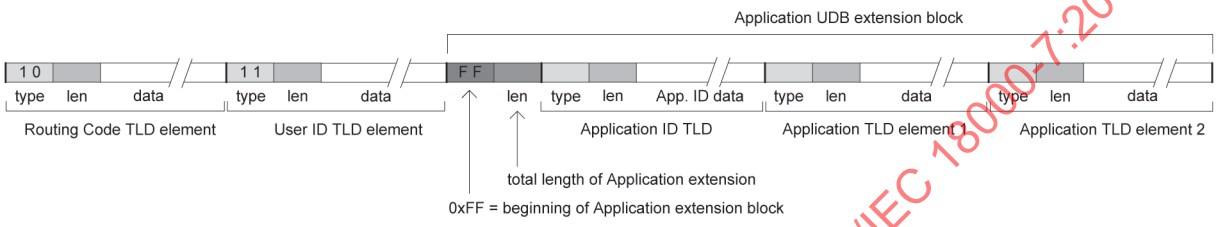


Figure 32 — Example Universal Data Block - UDB Type 0x00

7.4.4.4.2 Sleep

To put a tag to Sleep the command in Table 243 shall be sent (written) to the tag.

Table 243 — Sleep command format (Write)

Command code
0x15

Upon receiving the **Sleep** command in Table 243, the tag shall enter the Sleep state. The tag shall not respond to this command and shall ignore any subsequent commands until the tag is woken again by the Wake Up Signal.

7.4.4.4.3 Sleep all but

To put all except one tag to Sleep the command in Table 244 shall be sent (written) to the tag.

Table 244 — Sleep all but command format (Write)

Command code	Tag Manufacturer ID	Tag Serial Number
0x16	2 bytes	4 bytes

— **Tag Manufacturer ID**: the Tag Manufacturer ID of the tag which should remain awake following the Sleep All But command.

— **Tag Serial Number:** the Tag Serial Number of the tag which should remain awake following the Sleep All But command.

The Sleep All But command is a broadcast command used to place all tags into the sleep state, as with the Sleep command of section 6.3.2, except for the one tag that matches the provided Tag Manufactures ID and Tag Serial Number. Upon receiving this command, all tags except the one tag that matches the provided Tag Manufactures ID and Tag Serial Number shall enter “sleep” state.

The tags shall not respond to this command.

7.4.4.4.4 Security commands

Access to tag write commands shall be guarded by a *password protection* mechanism that application software can command the tag to engage or disengage (see Figure 4). If password protection is engaged, those write commands shall be *non-accessible unless the tag is unlocked*; that is, they will not perform their usual operations but rather respond with an Authorization Failure error. If the password protection is disengaged, the commands are *accessible* – they behave as described in the corresponding sections of this part of ISO/IEC 18000 Extended Mode without the possibility of an Authorization Failure error. Password protection is engaged and disengaged by means of the Set Password Protect Mode command described in section 6.3.4.2. Password protection is disengaged by default.

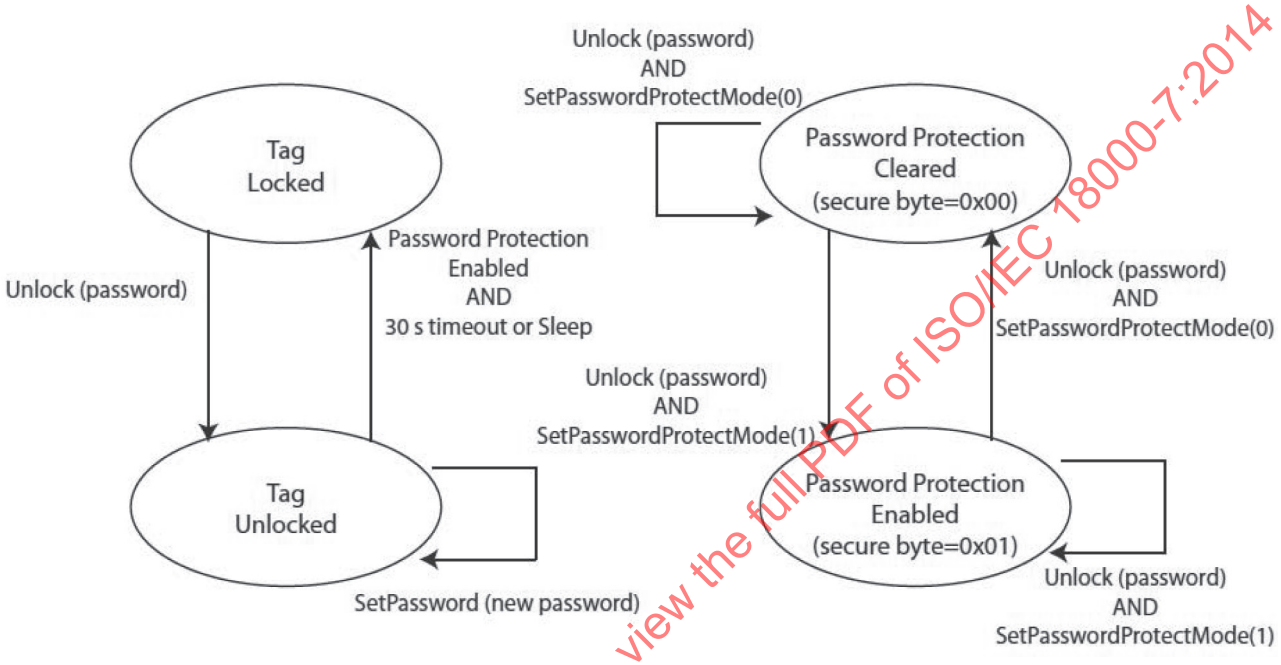


Figure 33 — Tag security state machines

While password protection is engaged, application software can command the tag to enter the *unlocked* state temporarily. While a tag is unlocked, the password-protected write commands shall be accessible. Any time the tag enters the sleep state (either the tag receives a “Sleep” or “Sleep All But” command or 30 seconds passes since the last well-formed command has been received), the tag shall return to the *locked* state, in which the password-protected commands shall be non-accessible. The Unlock command of section 6.3.4.3 puts the tag into the unlocked state. There is no command to put a tag into the locked state explicitly.

Table 245 lists the commands that are affected by password protection.

Table 245 — Write commands affected by password protection

Command code	Command name	Description
0x93	User ID	Sets user assigned ID (1 – 60 bytes)
0x89	Routing Code	Writes routing code
0xE0	Write Memory	Writes user memory
0x95*	Set Password*	Sets tag password (4 bytes long)
0x97*	Set Password Protect Mode*	Engages/disengages password protection
0x26	Table Create	Creates a database table
0x26	Table Add Records	Prepares to add new records to the specified database table

Table 245 *(continued)*

Command code	Command name	Description
0x26	Table Update Records	Prepares to modify the specified table records
0x26	Table Update Fields	Prepares to update the specified fields of a table record
0x26	Table Delete Record	Deletes existing record from the existing database table
0x26	Table Write Fragment	Writes a block of data into a table as initiated by the Table Add Records, Table Update Records, or Table Update fields command
0x8E	Delete Writeable Data	Deletes all allocated writeable data on a tag,

* These commands behave as though password protection were engaged permanently.

7.4.4.4.4.1 Security — Set Password

To set the password of a tag, the command in Table 246 shall be sent (written) to the tag.

Table 246 — Set Password command format (write)

Command code	Password
0x95	4 bytes

— **Password:** a four byte binary value, which shall act as the password for subsequent security commands.

To the Set Password command the tag shall respond with a point-to-point response message (and no data, unless an error is encountered) as shown in Table 247.

Table 247 — Set Password command format (write response)

Command code
0x95

This command sets the tag's password. This command requires tag to be first unlocked with the Unlock command of section 6.3.4.3 before the command can be accessed. The initial value of the tag's password is 0xFFFFFFFF.

The possible error responses shall be as shown in Table 248.

Table 248 — Set Password command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	**Password** parameter is missing or the wrong length
0x08	Authorization Failure	**Unlock** command not invoked prior to invocation of this command

7.4.4.4.4.2 Security — Set Password Protect Mode

To set a tag's Password Protect Mode the command in Table 249 shall be sent (written) to the tag.

Table 249 — Set Password Protect Mode command format (write)

Command code	Secure
0x97	1 byte

— **Secure:** a flag that specifies whether password protection shall be engaged or disengaged. The value 0x01 shall cause password protection to be engaged, the value 0x00 shall cause password protection to be disengaged.

To the Set Password Protect Mode command the tag shall respond with a point-to-point response message (and no data, unless an error is encountered) as shown in Table 250.

Table 250 — Set password Protect Mode command format (write response)

Command code
0x97

This command engages or disengages password protection in the tag. To access this command the tag shall first be unlocked with the Unlock command of section 6.3.4.3 regardless of the state of the tag's password protection.

The possible error responses shall be as shown in Table 251.

Table 251 — Set Password Protect Mode command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	Secure parameter is missing or the wrong length
0x08	Authorization Failure	Unlock command not invoked prior to invocation of this command

7.4.4.4.4.3 Security — Unlock

To unlock a tag the command in Table 252 shall be sent (written) to the tag.

Table 252 — Unlock command format (write)

Command code	Password
0x96	4 bytes

— **Password**: a four-byte binary value that was previously defined as the password via the Set Password command.

To the Unlock command the tag shall respond (write response) as shown in Table 253.

Table 253 — Unlock command format (write response)

Command code
0x96

This command unlocks the tag. If the supplied password matches tag's password, the tag shall permit the execution of all commands ordinarily non-accessible because of password protection. The tag shall remain in the unlocked state until it receives the Sleep command, Sleep All But command, or 30 seconds has elapsed since the tag received a command.

The possible error responses shall be as shown in Table 254.

Table 254 — Unlock command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	Password parameter is missing or the wrong length
0x08	Authorization Failure	Incorrect password supplied

7.4.4.4.5 Transit information commands

7.4.4.4.5.1 User ID

To retrieve a tag's User ID the command in Table 255 shall be sent to the tag.

Table 255 — User ID command format (read)

Command code
0x13

To the User ID read command the tag shall respond with a point-to-point response message with command code and data as shown in Table 256.

Table 256 — User ID command format (read response)

Command code	User ID Length	User ID
0x13	1 byte	N bytes

— **User ID Length:** the length in bytes of the User ID being returned, where N is between 0 and 60 inclusive.

— **User ID:** contents of the User ID on the tag.

To set a tag's User ID the command in Table 257 shall be sent to the tag.

Table 257 — User ID command format (write)

Command code	User ID Length	User ID
0x93	1 byte	N bytes

— **User ID Length:** the length, N, in bytes, of the User ID, where N is between 0 and 60 inclusive.

— **User ID**: the contents of the User ID

To the User ID write command the tag shall respond with a point-to-point response message with command code (and no data, unless an error is encountered) as shown in Table 258.

Table 258 — User ID command format (write response)

Command code
0x93

The User ID is a user-readable and writeable memory whose meaning and size (up to 60 bytes) is user defined. The User ID format and content shall follow the requirements of unique identifiers as defined in ISO/IEC 15459-3. Moreover, organisations wishing to allocate unique userids shall do so according to the rules defined by the accredited issuing agency. Issuing Agencies shall apply to the Registration Authority for registration according to 15459-2:

http://www.iso.org/iso/home/standards_development/list_of_iso_technical_committees/maintenance_agencies.htm

This command sets and gets the size and contents of the User ID. In addition to this command, the Collection with UDB and Read Universal Data Block commands also retrieve the User ID, except that when the User ID Length parameter is set to zero, the UDB message will not contain the User ID. The default length of the User ID is zero.

The possible error responses shall be as shown in Table 259.

Table 259 — User ID command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	The length of the User ID parameter does not agree with the User ID Length parameter, or the wrong number of parameter bytes was given, or the User ID Length parameter is greater than the maximum, 60
0x08	Authorization Failure	Write command was attempted with password protection engaged and tag in the locked state
0x0A	Operation Failed	Tag data corrupted, or internal failure on write of User ID on tag

7.4.4.4.5.2 Routing Code

To retrieve a tag's Routing Code the command in Table 260 shall be sent to the tag.

Table 260 — Routing Code command format (read)

Command code
0x09

To the Routing Code read command the tag shall respond with a point-to-point response message with command code and data as shown in Table 261.

Table 261 — Routing Code command format (read response)

Command code	Routing Code Length	Routing Code
0x09	1 byte	N bytes

— **Routing Code Length**: the length in bytes of the Routing Code being returned, where N is between 0 bytes and 50 bytes inclusive.

— **Routing Code**: contents of the Routing Code on the tag.

To set a tag's Routing Code the command in Table 262 shall be sent to the tag.

Table 262 — Routing Code command format (write)

Command code	Routing Code Length	Routing Code
0x89	1 byte	N bytes

— **Routing Code Length:** the length, N, in bytes, of the Routing Code, where N is between 0 and 50 inclusive

— **Routing Code:** the data to be written to Routing Code on the tag

To the Routing Code write command the tag shall respond with a point-to-point response message with command code (and no data, unless an error is encountered) as shown in Table 263.

Table 263 — Routing Code command format (write response)

Command code
0x89

The Routing Code is a user-readable and writable memory whose purpose and size (up to 50 bytes) is user defined. The Routing Code should be used as defined in the ISO 17363 standard. Note that the Routing Code is part of the tag's response to the Collection with UDB and Read Universal Data Block

commands, except that when the Routing Code Length parameter is set to zero, the UDB message will not contain the Routing Code. The default length of the Routing Code is zero.

The possible error responses shall be as shown in Table 264.

Table 264 — Routing Code command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	**Routing Code Length** parameter is greater than 50 (maximum length permitted), or the length of the **Routing Code** parameter does not agree with the **Routing Code Length** parameter, or the wrong number of parameter bytes was given
0x08	Authorization Failure	Write command was attempted with password protection engaged and tag in the locked state
0x0A	Operation Failed	Tag data corrupted, or internal failure on write of User ID on tag

7.4.4.4.6 Manufacturing Information Commands

The following two commands enable the tag manufacturer to provide manufacturer-defined, immutable information about a tag.

7.4.4.4.6.1 Firmware Version

To retrieve a tag's Firmware Version the command in Table 265 shall be sent to the tag.

Table 265 — Firmware Version command format(read)

Command code
0x0C

To the Firmware version command the tag shall respond with a point-to-point response message with command code and data as shown in Table 266.

Table 266 — Firmware Version command format (read response)

Command code	Firmware version
0x0C	4 bytes

— **Firmware Version:** tag firmware version from the tag, a manufacturer defined immutable value.

The Firmware Version indicates the tag firmware version.

7.4.4.4.6.2 Model Number

To retrieve a tag's Model Number the command in Table 267 shall be sent to the tag.

Table 267 — Model number command format (read)

Command code
0x0E

To the Model Number command the tag shall respond with a point-to-point response message with command code and data as shown in Table 268.

Table 268 — Model Number command format (read response)

Command code	Model number
0x0E	2 bytes

Model number: tag model number from the tag, a manufacturer defined immutable value.

The Model Number indicates the tag model number.

7.4.4.4.7 Memory commands

A tag may provide one or more bytes of user-readable and writable random-access memory in which the user can store and retrieve user-defined data. This memory is independent of all other data storage concepts (such as User ID and tables) defined in this part of ISO/IEC 18000-7 Extended Mode. Associated with every byte of memory is an unsigned integer *address*, through which that memory byte can be accessed. For B bytes of memory the addresses 0 through B-1 access the full range of memory.

7.4.4.4.7.1 Write Memory

To write memory the command in Table 269 shall be sent (written) to the tag.

Table 269 — Write Memory command format (write)

Command Code	Number of Bytes	Start Address	Data
0xE0	1 byte	3 bytes	N bytes

— **Number of Bytes:** N, the number of bytes to write, in the range 1 to 237 inclusive. The number of bytes of data in a Write Memory command message must be no greater than 255 – 18 = 237 (18 is the combined length of the command packet header, the number of bytes field, the start address field and the CRC bytes).

— **Start Address:** the memory address of the first memory byte to write, in the range 0 to the manufacturer-defined maximum address

— **Data:** the memory contents to write

To the Write Memory command the tag shall respond with a point-to-point response message with command code (and no data, unless an error is encountered) as shown in Table 270.

Table 270 — Write Memory command format (write response)

Command Code
0xE0

- The Write Memory command stores the given data into the user random-access memory for later retrieval with the Read Memory command of the next section.

The possible error responses shall be as shown in Table 271.

Table 271 — Write Memory command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	length of the Data parameter does not agree with the Number of Bytes parameter, or the wrong number of parameter bytes was given, or the Number of Bytes parameter is outside its legal range, or the Start Address plus Number of Bytes extends beyond the maximum address

Table 271 *(continued)*

Error Code	Error Name	Reason
0x08	Authorization Failure	Write command was attempted with password protection engaged and tag in the locked state
0x0A	Operation Failed	Tag data corrupted, or internal failure on write of memory on tag

7.4.4.4.7.2 Read Memory

To read memory the command in Table 272 shall be sent to the tag.

Table 272 — Read Memory command format (read)

Command Code	Number of Bytes to Read	Start Address
0x60	1 byte	3 bytes

— **Number of Bytes to Read:** the number of bytes to read, in the range 1 to 239 inclusive. The number of bytes of data in a Read Memory command message must be no greater than 255 – 16 = 239 (16 is the combined length of the response packet header, the number of bytes field and the CRC bytes).

— **Start Address:** the memory address of the first memory byte to read, in the range 0 to the manufacturer-defined maximum address

To the Read Memory command, the tag shall respond with a point-to-point response message with command code, parameter, and data as shown in Table 273.

Table 273 — Read Memory command format (read response)

Command Code	Number of Bytes Actually Read	Data
0x60	1 byte	N bytes

— **Number of Bytes Actually Read:** N, the number of bytes of data returned in the response, which always agrees with Number of Bytes to Read

— **Data:** the memory contents read from tag memory

- The Read Memory command retrieves from the user random-access memory the requested data previously written with the Write Memory command of the previous section.

The possible error responses shall be as shown in Table 274.

Table 274 — Read Memory command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	The wrong number of parameter bytes was given, or the Number of Bytes to Read parameter is outside its valid range, or the Start Address plus Number of Bytes to Read extends beyond the maximum address
0x0A	Operation Failed	Tag data corrupted, or internal failure on read of memory on tag

7.4.4.4.8 Delete Writeable Data

To delete all allocated writeable data on a tag, the command in Table 275 shall be sent to the tag. Data that is permanent on the tag and that is marked non-writeable is left untouched.

Table 275 — Delete Writeable Data

Command code
0x8E

To the Delete Writeable Data command the tag shall respond with a point-to-point response message with command code (and no data, unless an error is encountered) as shown in Table 276.

Table 276 — Delete Writeable Data (response)

Command code
0x8E

This command restores all user-writeable memory to factory defaults. In particular, the following operations are performed:

— The length of the User ID is reset to zero.

— The length of the Routing Code is reset to zero.

— All user database tables are deleted.

— The password shall be reset to 0xFFFFFFFF (initial value).

— Password Protect Mode is reset to disabled mode.

— Any existing database table tokens shall be invalidated.

— The Table Query Results table (Table 0x0000) shall be cleared.

The possible error responses shall be as shown in Table 277.

Table 277 — Delete Writeable Data command errors

Error Code	Error Name	Reason
0x08	Authorization Failure	Command was attempted with password protection engaged and tag in the locked state
0x0A	Operation Failed	Internal failure on deleting data on tag

7.4.4.4.9 Read Universal Data Block

The Read Universal Data Block command is used to read the Universal Data Block (UDB). The UDB can become large enough to require multiple Read Universal Data Block commands to retrieve the entire UDB. The Offset into UDB field allows an interrogator to retrieve a specific portion of the complete Universal Data Block. To read the Universal Data Block the Read UDB command in Table 278 shall be sent to the tag.

Table 278 — Read UDB

Command Code	UDB Type Code	Offset into UDB	Max Packet Length
0x70	1 byte	2 byte	1 byte

— **UDB Type Code:** identifies the requested UDB type.

— **Offset into UDB:** used by the interrogator to identify a starting offset into the specified UDB. In order to retrieve longer Universal Data Blocks, the interrogator will use multiple Read UDB commands and advance the offset value appropriately after each successfully received tag response

— **Max Packet Length:** an integer in the range 21 to 255 inclusive that specifies the maximum value that a tag can use as the Packet Length field in its response. The value 21 includes the 15 bytes of response packet wrapper, one byte of UDB Type Code, two bytes of Total UDB Length value, 2 bytes for the Requested Offset value and at least one byte of UDB data.

To the Read Universal Data Block command, the tag shall respond with a point-to-point response message with command code, parameters, and data as shown in Table 279.

Table 279 — Read UDB Response

Command Code	UDB Type Code	Total UDB Length	Requested Offset	Universal Data Block
0x70	1 byte	2 bytes	2 bytes	N bytes

— **UDB Type Code**: identifies the requested UDB type.

— **Total UDB Length:** the total length, in bytes, of UDB data on the tag for the selected UDB Type.

— **Requested Offset:** the value provided in the Interrogator's command message.

— **Universal Data Block:** a portion of the Universal Data Block.

To read the entire UDB, an Interrogator will begin with Offset into UDB set to 0 and Max Packet Length set to the largest acceptable packet size. Tags may select a smaller packet size than the length specified by Maximum Packet Length but may not exceed that value. After successfully receiving the initial portion of the UDB, the Interrogator may continue by advancing the Offset into UDB value to the next unread data byte position and sending a second Read UDB command. The interrogator may continue to read the entire UDB but that it does not have to read the entire UDB.

An Interrogator is not required to retrieve the entire UDB. In addition, the Interrogator is not restricted to send Read UDB commands with any ordered sequence of Offset into UDB values to the tag.

The possible error responses shall be as shown in Table 280.

Table 280 — Read UDB command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	The Offset into UDB parameter is greater than the total length of the specified UDB, or Max Packet Length is less than 21, or the wrong number of parameters bytes was given.

7.4.4.4.10 Database table commands

The Database Table commands provide basic database functionality, allowing application software to create one or more tables of varying schemas, perform table updates, and query a table. The Database Table commands provide no mechanism for performing table joins. The schema and maximum number of records of a Database Table is fixed at table creation time.

A table schema consists of a list of field (column) widths, in bytes. Fields are numbered (indexed) sequentially, left to right, starting at 0 for the first field. Every field in a table is untyped; that is, all field value comparisons are performed on a byte-for-byte basis, with equality being established between two fields if all bytes in each field match. One field is considered "less than" a second field if for some byte position *p* in the two fields, all bytes in the byte range 0 to *p*-1 are equal in the two fields, and byte *p* of the first field is less than byte *p* of the second field. In other words, a straight multi-byte value comparison is performed with the first byte being the most significant and the last byte being the least significant.

Table records (rows) are indexed starting at 0 for the first record. The record number (the record index) does not maintain a fixed relationship with a record. When a record is deleted, any remaining records in the database table are re-numbered and may be different than the record order prior to the Table Delete Record command.

Associated with a database table is a table ID, an immutable 2-byte value that is assigned at table creation time which uniquely identifies a table among all other tables in the tag.

The database tables can be divided into the following types by Table ID, as shown in Table 281.

Table 281 — Table ID space definitions

Table ID range	Table Type
0x0000 - 0x7FFF	ISO defined
0x8000 - 0xBFFF	Solution
0xC000 - 0xFFFF	Manufacturer / Vendor

Table IDs in the "ISO Defined" range are reserved for future inclusion in this part of ISO/IEC 18000 Extended Mode. Table ID 0x0000 is reserved for the Query Results table.

Table IDs in the "Solution" range are reserved for special features, functions and enhancements. In this region, the database tables are read and written with standard database commands, but the tables may have special functions and can have side effects. Table IDs within the Solution range must have published interfaces, and Table ID numbers shall be defined and assigned by the entity that owns the routing code.

Table IDs in the "Manufacturer/Vendor" range are reserved for vendor proprietary extensions, features and enhancements. In this region, the database tables are read and written with standard database commands, but the tables my have special functionality and can have side effects. Table IDs within the Manufacturer/Vendor range are available for use solely at the vendor's discretion, with no requirements to make public the purpose or use of the interfaces within this Table ID space. Data collected from a "Collect with UDB" command contains data from both the Manufacturers Data Block (MDB) and the Universal Data Block (UDB). The MDB data shall be stored in database tables within the range of the Manufacturer/Vendor Table ID space.

Read and Write Tokens

Certain table read and write commands produce a data element called a "token". Tokens provide a way for the tag implementation to abstract sequential data access to data sets larger than may be passed in a single message, and do some amount of error detection and recovery. The write commands (Table Add Records, Table Update Records, and Table Update Fields) declare a start location in logical terms (table ID, record #, field #) and a count. The read command, Table Get Data, declares only a start location. Upon receipt of one of these commands the tag generates a token value and returns it to the interrogator. Subsequently, the token is passed in a Table Read Fragment or Table Write Fragment command from the interrogator, back to the same tag, along with any necessary data (subject to context-dependent size restrictions). The tag then performs the read or write, also subject to context-dependent size restrictions, and generates a new token value. The new token is passed back to the interrogator for use in next Table Read Fragment or Table Write Fragment command.

The value of the token is completely at the discretion of the tag implementer, except for the following two requirements.

4. While the interrogator is issuing a series of Table Read Fragment or Table Write Fragment commands, by inspecting the token value the tag shall be able to differentiate the next command in the series from the most recently received command in that series. For example, if an interrogator sends the tag a command to read or write a fragment of data, receives no response from the tag, and then sends the same command again with the intention of reading or writing the same fragment, the tag shall identify it as a retry attempt (by means of the token). See Special Database Retry Situations section below.

5. In response to the last command of the series, as determined by the limits imposed by the Table Add Records, Table Update Records, Table Update Fields, or Table Get Data command that preceded the series, the tag shall return a single-byte token whose value is specifically 0x00. That special value informs the interrogator that the tag considers the series to be complete.

6. A tag shall support the existence of multiple, independent "read tokens," and may support the existence of multiple, independent write tokens. A tag shall support a minimum of two independent read tokens.

A "read token" is a token generated by an invocation of the Table Get Data command and used subsequently in invocations of the Table Read Fragment command. A "write token" is a token generated by an invocation of one of the table write commands and used subsequently in invocations of the Table Write Fragment command. The table write commands are Table Add Records, Table Update Records, and Table Update Record Fields. Supporting multiple, independent read tokens means that an invocation of Table Get Data or Table Read Fragment using one token does not affect the operation of those commands using another token, even if the two tokens are associated with the same table. Supporting multiple, independent write tokens means that an invocation of a Table Write command (Table Add Records, Table Update Records, and Table Update Fields) with one token shall not affect the operation of any other Table Write command with another token, provided that the two tokens are associated with different tables. However, invoking a table write command on a table will invalidate all read and write tokens associated with that table.

The high-order 4 bits of the first byte of the token indicates the length of the token, not including the first byte, so zero indicates a token length of 1 byte (see Table Write Fragment). The Token value of exactly 0x00 is reserved, and indicates an end-of-iteration condition. The structure of a Token field is shown below in Table 282:

Table 282 — Token structure

N: Token Length	Token Data
N value in bits 7-4 [Value of N = 0 – 15]	4 low order bits of Token Length byte, then N bytes

Table commands are categorized as being either a *read* command or a *write* command. The read commands include Table Get Data, Table Get Properties, Table Query, and Table Read Fragment, while the table write commands include Table Create, Table Add Records, Table Update Records, Table Update Fields, Table Delete Record, and Table Write Fragment. For all table write commands, the application will have to rewrite the data on the tag for any error occurs during the table write command operation.

Special Database Retry Situations

For the commands Table Add Records, Table Delete Records, Table Read Fragment, and Table Write Fragment special error handling is necessary if the interrogator does not receive the response from a successful completion of the command and, therefore, must do a retry of the command. A retry of the command shall be shall an identical copy of the initial invoked command packet, explicitly using the same Session ID, Command Code, Sub Command Code, Sequence ID or Request Token, Table ID (if used), and Data (if used) as the original. The tag shall determine if a command request is a retry of the previously successful database command by comparing it to the previously received command packet. If the tag identifies a request to be a retry of the previous executed and successful database command then the tag SHALL resend the same response from the previous successful command. Refer to command descriptions for Table Add Records, Table Delete Records, Table Read Fragment, and Table Write Fragment for additional details. Note that other database commands also may incur retry requests and retries should be supported.

7.4.4.4.10.1 Table Create

When invoking Table Create the command in Table 283 shall be sent to the tag.

Table 283 — Table Create

Command Code	Sub Command Code	Table ID	Maximum Number of Records	Number of Fields	Length of Each Field
0x26	0x01	2 bytes	2 bytes	1 byte	N bytes

Where:

— **Table ID** indicates the identifier to be assigned to the table. Valid ID range is 0x0001 to 0xFFFF. Table ID 0x0000 is reserved for the Query Results Table.

— **Maximum Number of Records** indicates how many records may ultimately exist in the table in total. Valid range is from 0x0001 to 0xFFFF. The remaining amount of unallocated table memory on the tag may additionally limit the valid range.

— **Number of Fields** the number of the fields, N, per record. Its valid range is 1 to 32

— **Length of Each Field** is a byte array of length N bytes. Each one-byte element of the byte array indicates the size of a field. The first element of the byte array specifies the length of the first field (index 0), the second element specifies the length of the second field (index 1), and so forth. The length of a field shall lie within the range 1 to 255 inclusive.

To the Table Create command the tag shall respond with a point-to-point response message with command code (and no data, unless an error is encountered) as shown in Table 284.

Table 284 — Table Create response

Command Code
0x26

This command creates a database table with a defined maximum number of records, the record format consisting of a specified number of fields each having a specified length. Initially after creation, the table has no records.

The possible error responses shall be as shown in Table 285.

Note If the tag identifies a request of this command to be a retry of the previous command that was executed successfully the tag SHALL NOT execute the request and instead, SHALL resend the same response from the previous successful command.

Table 285 — Table Create command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	A parameter is missing, or the Number of Fields parameter is outside its valid range, or the length of the Length of Field array does not match Number of Fields, or one or more of the Length of Field elements is zero, or the wrong number of parameter bytes was given.
0x06	Can't Create Object	The Table ID is already assigned to an existing table and this is not a retry command, or the tag does not have sufficient memory, or the Table ID is 0x0000. The following sub-codes define the kind of error: 0x02 Object already exists 0x03 Out of Memory 0x04 Reserved
0x08	Authorization Failure	Command was attempted with password protection engaged and tag in the locked state
0x0A	Operation Failed	Database is corrupted, or unable to create table regardless of valid command parameters or available memory

7.4.4.4.10.2 Table Add Records

When invoking Table Add Records the command in Table 286 shall be sent to the tag.

Table 286 — Table Add Records

Command Code	Sub Command Code	Table ID	Sequence ID	Number of Records
0x26	0x02	2 bytes	1 byte	2 bytes

Where:

— **Table ID** indicates the identifier assigned to the table.

— **Sequence ID** is used to identify unique transactions. For each invocation of this command, the interrogator shall supply a different value for Sequence ID. If the interrogator receives no reply from an invocation of the command (due to a communication error, for example) the interrogator shall retry the Table Add Record command using the same Sequence ID as the unsuccessful attempt. The tag shall verify the Sequence ID is different from the value provided with the last successful Table Add Record command, only then is the table record added.

— **Number of Records** indicates the total number of records to add to the table. Valid range is 1 to the Maximum Number of Records set at the time of table creation minus the number of records previously added to the table.

To the Table Add Records command the tag shall respond with a point-to-point response message with command code and data as shown in Table 287.

Table 287 — Table Add Records

Command Code	Token
0x26	N bytes

— Token indicates a value used to iteratively write data to the added records. The Token value of exactly 0x00 is reserved, and indicates an end-of-iteration condition. The structure of a Token field is shown above in Table 82:

This command instructs the tag to prepare to add the specified number of records to the Table. The record contents are written to the table with a sequence of Table Write Fragment commands. This command invalidates any existing tokens for this Table ID. This command also invalidates any Table Query results present in Table 0x0000.

NOTE If the tag identifies a request of this command to be a retry of the previous command that was executed successfully the tag SHALL NOT execute the request and instead, SHALL resend the same response from the previous successful command.

The possible error responses shall be as shown in Table 288.

Table 288 — Table Add Records command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	Number of Records is zero, or the wrong number of parameter bytes was given, or the Sequence ID is the same value used with the previous same command.
0x04	Not Found	There is no database table associated with the specified table ID
0x08	Authorization Failure	Command was attempted with password protection engaged and tag in the locked state
0x09	Object is Read-Only	Table ID is 0x0000, which specifies the read-only query results table
0x41	Boundary Exceeded	The table is too full to accept an additional Number of Records new records

7.4.4.4.10.3 Table Update Records

When invoking Table Update Records the command in Table 289 shall be sent to the tag.

Table 289 — Table Update Records

Command Code	Sub Command Code	Table ID	Starting Record Number	Number of Records
0x26	0x03	2 bytes	2 bytes	2 bytes

Where:

— **Table ID** indicates the identifier assigned to the table.

— **Starting Record Number** indicates the first record to begin updating. Valid range is 0 up to (Number of Records in the Table - 1).

— **Number of Records** indicates the total number of records that will be updated. Valid range is 1 up to (Number of Records in the Table - Starting Record Number).

To the Table Update Records command the tag shall respond with a point-to-point response message with command code and data as shown in Table 290.

Table 290 — Table Update Records response

Command Code	Token
0x26	N bytes

— Token indicates a value used to iteratively write data to the updated records. The Token value of exactly 0x00 is reserved, and indicates an end-of-iteration condition. The structure of a Token field is shown in Table 82.

This command instructs the tag to prepare to update the specified table records. The new record contents are written to the table with a sequence of Table Write Fragment commands. This command invalidates any existing tokens for this Table ID. This command also invalidates any Table Query results present in Table 0x0000.

The possible error responses shall be as shown in Table 291.

Table 291 — Table Update Records command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	**Number of Records** is zero, or the wrong number of parameter bytes was given
0x04	Not Found	There is no database table associated with the specified **table ID**
0x08	Authorization Failure	Command was attempted with password protection engaged and tag in the locked state
0x09	Object is Read-Only	**Table ID** is 0x0000, which specifies the read-only query results table
0x41	Boundary Exceeded	Starting **Record Number plus Number of Records** extends beyond the total number of records in the table

7.4.4.4.10.4 Table Update Fields

When invoking Table Update Fields the command in Table 292 shall be sent to the tag.

Table 292 — Table Update Fields

Command Code	Sub Command Code	Table ID	Record Number	Starting Field Number	Number of Fields
0x26	0x04	2 bytes	2 bytes	1 byte	1 byte

Where:

— **Table ID** indicates the identifier assigned to the table.

— **Record Number** indicates the record to update. Valid range is 0 up to (Number of Records in the Table - 1).

— **Starting Field Number** indicates the first field to begin updating. Valid range is from 0 up to (Number of Fields in the Table - 1).

— **Number of Fields** indicates the total number of fields in the specified record that will be updated. Valid range is 1 up to (Number of Fields in the Table - Starting Field Number).

This command instructs the tag to prepare to update the specified fields of a table record. The new field contents are written with a sequence of Table Write Fragment commands. This command can only modify fields within a single record, which is provided as the Record Number. This command invalidates any existing tokens for this Table ID. This command also invalidates any Table Query results present in Table 0x0000.

To the **Table Update Fields** command the tag shall respond with a point-to-point response message with command code and data as shown in Table 293.

Table 293 — Table Update Fields

Command Code	Token
0x26	N bytes

— Token indicates a value used to iteratively write data to the updated records. The Token value of exactly 0x00 is reserved, and indicates an end-of-iteration condition. The structure of a Token field is shown in Table 82.

The possible error responses shall be as shown in Table 294.

Table 294 — Table Update Fields command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	Number of Fields is zero, or the wrong number of parameter bytes was given
0x04	Not Found	There is no database table associated with the specified table ID
0x08	Authorization Failure	Command was attempted with password protection engaged and tag in the locked state
0x09	Object is Read-Only	Table ID is 0x0000, which specifies the read-only query results table
0x41	Boundary Exceeded	Record Number is greater than or equal to the total number of records in the table, or Number of Fields plus Starting Field Number extends beyond the number of fields in the table

7.4.4.4.10.5 Table Delete Record

When invoking Table Delete Record the command in Table 295 shall be sent to the tag.

Table 295 — Table Delete Record

Command Code	Sub Command Code	Table ID	Sequence ID	Record Number
0x26	0x05	2 bytes	1 byte	2 bytes

Where:

— **Table ID** indicates the identifier assigned to the table.

— **Sequence ID** is used to identify unique transactions. For each invocation of this command, the interrogator shall supply a different value for Sequence ID. If the interrogator receives no reply from an invocation of the command (due to a communication error, for example) the interrogator shall retry the Table Delete Record command using the same Sequence ID as the unsuccessful attempt. The tag shall verify the Sequence ID is different from the value provided with the last successful Table Delete Record command, only then is the table record deleted.

— **Record Number** indicates the index number of the record to delete

To the Table Delete Record command the tag shall respond with a point-to-point response message with command code (and no data, unless an error is encountered) as shown in Table 296.

Table 296 — Table Delete Record

Command Code
0x26

This command instructs the tag to delete a single record from the Table, renumbering the record numbers of the remaining records in such a way as to keep the record numbers contiguous starting with 0x0000 (zero). Following execution of Table Delete Record, the order of the remaining records in the table is undefined, and may be different than the record order prior to the Table Delete Record command.

This command invalidates any existing tokens for this Table ID. To read or write data to the database, a new table write command (Table Add Records, Table Update Records, Table Update Fields) or table read command (Table Get Data) shall be issued.

This command also invalidates any Table Query results present in Table 0x0000.

The possible error responses shall be as shown in Table 228.

If the tag identifies a request of this command to be a retry of the previous command that was executed successfully the tag SHALL NOT execute the request and instead, SHALL resend the same response from the previous successful command.

Table 297 — Table Delete Record command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	The wrong number of parameter bytes was given or the Sequence ID is the same value used with the previous same command.
0x04	Not Found	There is no database table associated with the specified table ID
0x08	Authorization Failure	Command was attempted with password protection engaged and tag in the locked state
0x09	Object is Read-Only	Table ID is 0x0000, which specifies the read-only query results table
0x0A	Operation Failed	Database is corrupted, or unable to complete record removal
0x41	Boundary Exceeded	Record Number is greater than or equal to the total number of records in the table

7.4.4.4.10.6 Table Get Data

When invoking **Table Get Data** the command in Table 298 shall be sent to the tag.

Table 298 — Table Get Data

Command Code	Sub Command Code	Table ID	Starting Record Number	Starting Field Number
0x26	0x06	2 bytes	2 bytes	1 byte

Where:

— **Table ID** indicates the identifier assigned to the table.

— **Starting Record Number** indicates the first record to begin reading.

— **Starting Field Number** indicates the first field to begin reading.

To the Table Get Data command the tag shall respond with a point-to-point response message with command code and data as shown in Table 299.

Table 299 — Table Get Data response

Command Code	Token
0x26	N bytes

Where:

— Token indicates a value used to iteratively read record data. The Token value of exactly 0x00 is reserved, and indicates an end-of-iteration condition. The structure of a Token field is shown in Table 82.

The Table Get Data command instructs the tag to prepare to read data from a database table starting with a specified record and field. A sequence of Table Read Fragment commands performs the actual data reading. Unlike the table write commands Table Add Records, Table Update Records, Table Update Fields, and Table Get Data is an open-ended iteration that terminates either at the application software's choosing or when the end of the table is reached.

The Table Get Data tokens, and subsequent tokens returned by Table Read Fragment are invalidated by any of the following commands: Delete Writeable Data, Table Add Records, Table Update Records, Table Update Fields, Table Delete Record and Table Write Fragment, which operate on the same Table ID.

The possible error responses shall be as shown in Table 300.

Table 300 — Table Get Data command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	The wrong number of parameter bytes was given
0x04	Not Found	There is no database table associated with the specified table ID, or Table ID is 0x0000 and there is no query result, either because no query was executed or the query result has been made invalid by an intervening table write command on the table that was queried or by starting a new query.
0x41	Boundary Exceeded	Starting Record Number is greater than or equal to the total number of records in the table, or Starting Field Number is greater than or equal to the number of fields in the table

7.4.4.4.10.7 Table Get Properties

When invoking Table Get Properties the command in Table 301 shall be sent to the tag.

Table 301 — Table Get Properties

Command Code	Sub Command Code	Table ID
0x26	0x07	2 bytes

Where:

— **Table ID** indicates the identifier assigned to the table.

The Table Get Properties command retrieves information about the specified table. It retrieves the number of used (filled) records in the table and the maximum number of records defined for the table.

To the Table Get Properties command the tag shall respond as shown in Table 302.

Table 302 — Table Get Properties response

Command Code	Total Number of Records	Maximum Number of Records	Reserved
0x26	2 bytes	2 bytes	1 bytes

Where:

— **Total Number of Records** indicates total number of records in the table.

— **Maximum Number of Records** indicates the maximum number of records specified for the table as specified at table creation by the Table Create command.

— **Reserved** is a byte reserved for future use and shall have the value 0x00.

The possible error responses shall be as shown in Table 303.

Table 303 — Table Get Properties command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	The wrong number of parameter bytes was given
0x04	Not Found	There is no database table associated with the specified table ID

7.4.4.4.10.8 Table Read Fragment

When invoking Table Read Fragment the command in Table 304 shall be sent to the tag.

Table 304 — Table Read Fragment

Command Code	Sub Command Code	Request Token	Requested Read Length
0x26	0x08	N bytes	1 byte

Where:

— **Request Token** is the token from the prior Table Get Data or Table Read Fragment command. The Token value of exactly 0x00 is reserved, and indicates an end-of-iteration condition. The structure of a Token field is shown in Table 282.

— **Requested Read Length** is the requested length of data to return. Valid range is from 1 to 46 bytes.

To the Table Read Fragment command the tag shall respond with a point-to-point response message with command code and data as shown in Table 305.

Table 305 — Table Read Fragment response

Command Code	Response Token	Actual Read Length	Data
0x26	N bytes	1 byte	M bytes

Where:

— **Response Token** is the resulting new token from a successful Table Read Fragment command. The Token value of exactly 0x00 is reserved, and indicates an end-of-iteration condition.

— **Actual Read Length** is the number of bytes of data actually read, and may be less than or equal to Requested Read Length.

— **Data** is the actual data read from the tag database table and is the Actual Read Length bytes long

The Table Read Fragment command reads a block of data bytes from a database table. The database table contents to be read are inherently identified by the Request Token received from the tag via a prior invocation of the Table Get Data command or a previous invocation of this Table Read Fragment command.

The Table Read Fragment command cannot read beyond the last record of a table. If the initial byte to be read by the Table Read Fragment command is within the table, but the Requested Read Length would reach beyond the end of the last record in the table, the command shall be considered valid, and shall return as Actual Read Length not more than the number of bytes remaining to be read in the table.

The possible error responses shall be as shown in Table 306.

NOTE If the tag identifies a request of this command to be a retry of the previous command that was executed successfully the tag SHALL resend the same response from the previous successful command.

Table 306 — Table Read Fragment command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	Request Token is malformed (as defined by the tag implementation), or Requested Read Length is zero, or the wrong number of parameter bytes was given. Request Token is malformed (as defined by the tag implementation), or Requested Read Length is zero, or the wrong number of parameter bytes was given, or the Request Token is 0x00
0x40	Stale Token	Request Token is properly formed and not 0x00 but is invalid due to an intervening modification of the table(s) to which the Request Token applies. These modifications include invocations of the following commands, associated with the Table ID supplied to the commands: Table Add Records, Table Update Records, Table Update Fields, and Table Delete Record.
0x0A	Operation Failed	Read operation failed or database is corrupted

7.4.4.4.10.9 Table Write Fragment

When invoking Table Write Fragment the command in Table 307 shall be sent to the tag.

Table 307 — Table Write Fragment

Command Code	Sub Command Code	Request Token	Data Length	Data
0x26	0x09	N bytes	1 byte	N bytes

Where:

— **Request Token** is the token from the prior Table Add Records, Table Update Records, Table Update Fields, or Table Write Fragment command. The structure of a Token field is shown in Table 82.

— **Data Length** is the length of data to write. Valid range is from 1 to 46 bytes.

— **Data** is the data bytes to be written to the tag database table.

To the Table Write Fragment command the tag shall respond with a point-to-point response message with command code and data as shown in Table 308.

Table 308 — Table Write Fragment response

Command Code	Response Token
0x26	N bytes

Where:

— **Response Token** is the resulting new token from a successful Table Write Fragment command. The Token value of exactly 0x00 is reserved, and indicates an end-of-iteration condition. The structure of a Token field is shown in Table 282.

The Table Write Fragment command writes a block of data bytes to a database table. The database table contents to write are inherently identified by the Request Token received from the tag via a prior invocation of the Table Add Records, Table Update Records, or Table Update Fields command or a previous invocation of this Table Write Fragment command.

This command invalidates any existing tokens for this Table ID. This command also invalidates any Table Query results present in Table ID 0x0000

The possible error responses shall be as shown in Table 309.

NOTE If the tag identifies a request of this command to be a retry of the previous command that was executed successfully the tag SHALL resend the same response from the previous successful command.

Table 309 — Table Write Fragment command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	Request Token is malformed (as defined by the tag implementation), or Data Length is zero, or the length of the Data parameter does not agree with Data Length, or the wrong number of parameter bytes was given, or the Request Token is 0x00
0x08	Authorization Failure	Command was attempted with password protection engaged and tag in the locked state
0x0A	Operation Failed	Write operation failed or database is corrupted
0x40	Stale Token	Request Token is properly formed and not 0x00 but is invalid due to an intervening modification of the table(s) to which the Request Token applies. These modifications include invocations of the following commands, associated with the Table ID supplied to the commands: Table Add Records, Table Update Records, Table Update Fields, and Table Delete Record.
0x41	Boundary Exceeded	The Data Length for this request would exceed the length declared in the original Table Add Records, Table Update Records, or Table Update Record Fields command

7.4.4.4.10.10 Table Query

When invoking Table Query the command in Table 310 shall be sent to the tag. The Table Query command can be sent as either a Broadcast message to all tags simultaneously, or as a Point-to-Point message to a single tag.

Table 310 — Table Query

Command Code	Sub Opcode	Table ID	Sequence ID	Query Element				
				Logical Operator	Logical Operand			
					Field Number	Relational Operator	Comparison Data Length	Comparison Data
0x26	0x10	2 bytes	1 byte	1 byte	1 byte	1 byte	1 byte	N bytes

Where:

— **Table ID** indicates the identifier assigned to the table.

— **Sequence ID** identifies a query element among a sequence of query elements. For a sequence of N query elements, the Sequence ID is N-1 for the first query element, N-2 for the second query element, and so forth, down to 0 (zero) for the Nth query element. The tag shall support a minimum of 4 query elements per sequence; Sequence IDs from 3 down to 0. The actual number of query elements supported on a tag can be retrieved through the UDB Element Type 0x15 (Table Query Size).

— **Logical Operator** defines the role of the current query element within the complete query. The possible values of the logical operator are the ISO 8859-1 characters 'C' (CLEAR), 'A' (AND), or 'O' (OR).

— **Field Number** indicates index number of the field to match. The Field Number shall be less than the number of fields in the table.

— **Relational Operator** defines the method by which the field contents are compared with Data. The possible values of the relation operator are the ISO 8859-1 characters '=' (EQUAL), '<' (LESS THAN), '>' (GREATER THAN), or '!' (NOT EQUAL).

— **Comparison Data Length** indicates length of the Comparison Data in bytes. The range of Comparison Data Length is 1 to 32.

— **Comparison Data** specifies the byte array to which the field contents are compared. Comparison Data is **Comparison Data Length** bytes long, and may include the special ISO/IEC 8859-1 prefix '*', the wildcard character.

Overview of Query Syntax

This command defines a *query element*, one table search criterion among a sequence of such criteria. A complete query conceptually has the form:

{<query element$_1$>} {<query element$_2$>} ... {<query element$_N$>}

where each <query element>, of which there is at least one, has the form:

<logical operator> <logical operand>

where <logical operand> has the form:

<field number> <relational operator> <comparison data>

where Logical Operator, Field Number, Relational Operator, and Comparison Data are fields in the Table Query command format shown in Table 110. Logical Operator is one of the ISO/IEC 8859-1 characters 'C', 'A', or 'O', Field Number is a table field, Relational Operator is one of the ISO/IEC 8859-1 characters '=', '<', '>', '!', and Comparison Data is a 1 to 32 byte string of data bytes. The angle brackets (<, >) and curly

braces ({, }) in the above syntax serve only as delimiters for the purposes of this discussion and have no syntactic meaning or literal presence in an actual command. A complete query, therefore, is specified as a sequence of Table Query commands.

Query Elements

Query elements within a complete query are related to one another by their logical operators, which specify how those query elements are aggregated into a compound Boolean expression. A logical operator can be a logical AND, a logical OR, or the special case CLEAR. Logicals AND and OR are left-associative binary operators of equal precedence which have their conventional Boolean meanings, while CLEAR merely indicates that the query element is the first element of the complete query. If CLEAR is the logical operator for any query element, any prior set of query elements are discarded and the current query element is to be regarded as the first query element of a new query. Upon receipt of a valid Query containing a CLEAR, any pre-existing results from any previous query shall be removed; all existing records in Table 0x00 shall be deleted.

The relational operands consist of the database table field identified by the Field Number and the Comparison Data.

Interpretation of Queries

A complete query is to be interpreted as an expression whose constituents are the logical operator and logical operand of each query element, read left to right. For example, suppose a complete query is composed of four query elements and the logical operands of the first, second, third, and fourth query elements are A, B, C, and D, respectively. The complete query

(CLEAR A) (AND B) (OR C) (AND D)

is to be interpreted as the Boolean expression

CLEAR (((A AND B) OR C) AND D)

where for each record of the table being searched each logical operand evaluates to “true” or “false” values as described below, the Boolean operators combine those values into a single “true” or “false” value in the conventional manner for Boolean operators, and the CLEAR operator has no impact on the Boolean value of the entire expression. If the entire expression evaluates to “true” the table record is included in the query results table, also described below.

Logical Operands

A logical operand specifies how each record of the table to be searched is to be checked for inclusion in the set of matching records. The field number of the logical operand specifies which field of each record is to be inspected. The comparison data specifies the value to which the field contents are to be compared. And the relational operator specifies the manner in which the field contents and comparison data, the two *relational operands* of the relational operator, are to be compared. Additionally, the first byte of the comparison data affects the nature of the comparison. If that byte is the ISO/IEC 8859-1 character ‘*’, the comparison is a *wildcard comparison*; otherwise, the comparison is a *full-match comparison*.

Full-Match Comparisons

If the relational operator is ‘=’ for a full-match comparison, the relational operands are compared on a byte-for-byte basis for an exact match. If the bytes at some position in both relational operands do not match, the logical operand evaluates to “false.” If one relational operand is longer than the other, the logical operand evaluates to “false.” Otherwise, the logical operand evaluates to “true.” For example, the following comparison evaluates to “true.”

“abc” ‘=’ “abc”

The following comparisons evaluate to “false.”

“abc” ‘<’ “abc”

“abdb” ‘<’ “abce”

“abc” ‘>’ “abc”

“abc” ‘!’ “abc”

If the **Relation Operator** is ‘!’ for a full-match comparison, the comparison is handled in the same manner as the ‘=’ relation operator but generates the opposite result. Any comparison in which the ‘=’ operator would result in the “true” condition, the ‘!’ operator results in “false,” and any comparison in which the ‘=’ operator would result in the “false” condition, the ‘!’ operator results in “true.” The following comparisons evaluate to “true.”

“abc” ‘!’ “abcd”

“abc” ‘!’ “ABC”

“abc” ‘!’ “abd”

“abc” ‘!’ “ab”

The following comparison results in “false.”

“abc” ‘!’ “abc”

If the Relational Operator is ‘<’ or ‘>’ for a full-match comparison, the relational operands are compared on a byte-for-byte basis as for the ‘=’ operator until the first non-matching byte is found. If no non-matching byte is found, the logical operand evaluates to “false.” The non-matching bytes are compared according to the relational operator. If the operator is ‘<’ and the byte from the field contents is less than the byte from the comparison data, the logical operand evaluates to “true.” If the operator is ‘>’ and the byte from the field contents is greater than the byte from the comparison data, the logical operand evaluates to “true.” For the inequality operators ‘<’ and ’>’, if the length of one relational operand is less than that of the other relational operand, then the shorter operand is considered for the purposes of comparison to contain an additional final byte whose value is less than the minimum possible value for a byte.

Note that because the comparison data is limited to 32 bytes, for fields greater than 32 bytes in length, full-match comparisons with the ‘=’ operator always evaluate to “false,” while full-match comparisons with the ‘!’ operator always evaluate to “true.”

For example, the following comparisons evaluate to “true.”

“abb” ‘<’ “abc”

“aad” ‘<’ “abc”

“ab” ‘<’ “abc”

“abc” ‘<’ “ad”

“abc” ‘>’ “abb”

“abc” ‘>’ “aad”

“abc” ‘>’ “ab”

“ad” ‘>’ “abc”

“abc” ‘!’ “abd”

“abc” ‘!’ “ab”

The following comparisons evaluate to “false.”

“abc”‘<’“abc”

“abdb”‘<’“abce”

“abc”‘>’“abc”

“abc” ‘!’ “abc”

Wildcard Comparisons

For wildcard comparisons with the relational operator ‘=’, the field contents starting with the first byte and the comparison data starting with the byte after ‘*’ are compared on a sliding basis for a match as in a full-match comparison until the end of the field contents is reached. That is, starting from the beginning of the field contents and sliding to the right a byte at a time until the end of the field contents, Comparison Data Length bytes of field data are compared against the Comparison Data Length bytes of Comparison Data bytes looking for a complete match. If a complete match is found, the comparison is discontinued. If a complete match was found and the Relational Operator was ‘=’, the comparison evaluates to a “true”, otherwise it evaluates to a “false”. The results for the ‘!’ operator are “false” if the complete match was found, otherwise it evaluates to a “true”. Wildcard comparisons with the relational operators ‘<’ and ‘>’ are illegal, as are wildcard comparisons for which the comparison data is the single character ‘*’.

In the following examples, the first item is the field number, and the last item is the Comparison Data.

The following comparisons evaluate to “true.”

“abcde” ‘=’ “*bcd”

“abcbcde” ‘=’“*bcd”

“abcecd” ‘!’ “*bcd”

The following comparisons evaluate to “false.”

“abcde” ‘!’ “*bcd”

“abce” ‘=’ “*bcd”

Query Failures

The possible error responses shall be as shown in Table 311.

Table 311 — Table Query command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	Sequence ID is greater than the maximum number of query operators that the tag supports; or Sequence ID is not the same as, or one less than, that of the previous Table Query command AND Logical Operator is not CLEAR; or Table ID is not the same as that of the previous Table Query command AND Logical Operator is not CLEAR; or Comparison Data Length, Logical Operator or Relational Operator are outside their valid range of values; or Data Length is zero; or the length of Data does not agree with Data Length, or the wrong number of parameter bytes was given
0x04	Not Found	There is no database table associated with the specified Table ID
0x41	Boundary Exceeded	Field Number greater than or equal to the number of fields in the table

Execution of Complete Query

Upon receipt of the final **Table Query** command (which has a Sequence ID of zero), the tag should now have the complete Query criteria. The tag shall execute the complete query on each record in the table

identified by Table ID, beginning with Record Number 0 and incrementing through all the records in the table.

The Query Results Table (Table ID 0x0000) contains the complete results of the query operation. The Query Results Table has records with a single two-byte field. Each 2-byte field/record in the Query Results table contains the record number of a matching record in the queried table. The matching record numbers, if any, in the Query Results table, shall increase monotonically. Records in Table 0x0000 shall be returned in response to Table Get Data and Table Read Fragment. The record number of each matching record shall be returned as individual records in MSB first order.

Point-to-Point and Broadcast Queries

The Table Query command exists as both a point-to-point command and a broadcast command. The value of the Packet Options field, determines whether the command is broadcast or point-to-point. The tag does not respond with any message to any broadcast Table Query command, even in case of error.

For non-final point-to-point Table Query commands, which have a non-zero Sequence ID, the tag shall verify it received a valid non-final Table Query command. If the command is a valid initial or intermediate Table Query command, the tag shall respond with a point-to-point response message with command code (and no data, unless an error is encountered) as shown in Table 312. This response merely indicates that the tag successfully received a valid query element, so no database query operation results are available or expected.

Table 312 — Intermediate Table Query Response

Command Code
0x26

Upon completion of the point-to-point query command sequence, the tag shall respond with a point-to-point response message with command code and data as shown in Table 313. If the query resulted in no records matched, the number of records matching the criteria shall be zero and the index of the first matched record shall be zero in response to the final point-to-point **Table Query** command.

Table 313 — Final Point-to-Point Intermediate Table Query response

Command Code	Number of Records matched	Index of first matched record
0x26	2 bytes	2 bytes

— **Number of Records Matched** contains the number of records found in the queried table, which meet the complete query criteria. This field shall be formatted as an unsigned 16-bit integer. If no matching records were found, this field shall contain 0.

— **Index of First Matched Record** contains the record number of the first matching record of the queried table, which meets the complete query criteria. If no records were found, this field shall contain 0.

An Interrogator may follow up a sequence of point-to-point Table Query commands by using the Table Get Data and Table Read Fragment commands to retrieve the results from the Query Results Table (Table 0x0000).

Broadcast Collection with UDB (Query Results UDB)

The broadcast Collection with UDB command may be used to retrieve the query results from tags after the complete sequence of Table Query commands has been transmitted. To retrieve the query results, an Interrogator may send the Collection with UDB command with the UDB Type field set to 0x02 (see Table 36). Tags will return their Tag serial number with the Table Query Results element (see Table 36). The Table Query Results element includes the index of the queried table, the number of matching records and the index of the first matching record. If the query resulted in no records matched, the number of

matching records shall be zero and the index of the first matched record shall be zero. The Interrogator may then follow up successful query matches by retrieving the records in the Query Results Table (Table 0x0000) with Table Get Data and Table Read Fragment commands.

Deleting Table Query Results

The results of the Table Query command are written to the query results table (reserved Table ID 0x0000). Any database command that modifies any of the database tables on the tag (Table Add Records, Table Update Records, Table Update Fields, Table Delete Record) shall force deletion of all records in the query results table. Any subsequent Table Get Properties commands for the query results table shall return 0 as the the number of records currently in the table.

7.4.4.4.11 Beep ON/OFF

When invoking Beep ON/OFF the command in Table 314 shall be sent to the tag.

Table 314 — Beep ON/OFF

Command Code	Beeper On/Off
0xE1	1 byte

Where:

— **Beeper On/Off** parameter when 0x01 will turn tag's beeper ON or when set to 0x00 will turn tag's beeper OFF.

To the Beep ON/OFF command the tag shall respond with a point-to-point response message with command code (and no data, unless an error is encountered) as shown in Table 315.

Table 315 — Beep ON/OFF response

Command Code
0xE1

The Beep ON/OFF command turns the tag's beeper on or off. When the tag's beeper is turned on, the beeper stays on until explicitly turned off or until the tag returns to the Sleep state.

The possible error responses shall be as shown in Table 316.

Table 316 — Beep ON/OFF command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	Beeper On/Off parameter is missing or the wrong length or outside its valid range of values

7.4.4.4.12 Sensor implementation

The retrieval and transmission of data and control information related to tag-based sensors can be implemented within this part of ISO/IEC 18000 Extended Mode. Sensor status and data can be added to a returned Universal Data Block using an Application extension block. Sensor data logs and control information can be read and written using the existing database table commands.

7.4.4.4.12.1 UDB application extensions for sensors

Manufacturers can record sensor status in the UDB using the UDB Application Extension Block format. Manufacturers can record sensor status in the UDB using the UDB Application Extension Block format.

Depending on the application and the complexity of the sensor implementation, the UDB application extension block provides flexibility on how to report sensor status. Specifically, one or many TLD elements can be defined in the UDB according to the TLD element format.

7.4.4.4.12.2 Sensor data storage

Sensor data and control information can be stored in database tables. Sensor related data can be stored in either the Solution Table ID space or the Manufacturer / vendor Table ID space depending on the application.

7.4.4.4.12.3 Commands to retrieve sensor status information and sensor data

The following standard commands can be triggered by the interrogator to retrieve sensor status information in the UDB:

— Collection with UDB (command code 0x1F)

— Read UDB (command code 0x70)

Sensor activity logs can be retrieved using the following commands

— Table Get Data (command code 0x26+0x06 followed by a Table Read Fragment (command code 0x26+0x08)

Or

— Table Query (command code 0x26+0x10)

Standard response message formats are sent back to the interrogator with the requested information.

7.4.4.4.12.4 Sensor data characteristics and formats

The stored data from the sensor should follow the formats described in IEEE 1451 and ISO/IEC 24753.

7.4.4.4.12.5 Physical interface between the sensor and RF tag

The physical interface between the sensor and the RF tag should be as described in IEEE 1451.7.

7.4.4.5 Tag collection (optional)

This standard specifies the optional method by which the interrogator shall identify and communicate with one or more tags present in the operating field of the interrogator over a common radio frequency channel. Communications specified include methods to: identify a tag, read data from a tag, write data to a tag and command the tag to perform a specific function. Tags do not transmit unless commanded to do so by the interrogator, and an interrogator can communicate with tags individually, or with the tag population as a whole.

In the following discussion, the terms all tags and tag population refer only to tags within the operating field of the interrogator.

The tag collection process is used to identify tags in the operating field of the interrogator. This is an iterative process that includes methods for coordinating responses from the tag population.

Tag collection procedure in the Extended Mode of operation differs from Base Mode since the MAC layer is IEEE 802.15.4 based. Depending on the MAC mode of operation (beacon or beaconless) the tags will follow MAC rules and create a network of devices with the interrogator. This procedure will be explained in this specification in more details. Once the network is formed, the application layer on the interrogator can exchange application layer commands such as a Collection with Universal Data Block, a Read Universal Data Block and a Sleep Command with the tags and collect the tag data.

Collection process may include following steps:

— Interrogator: transmits the Wake Up Signal to all tags in the operating field of the interrogator

— Tag: wake up into the ready state and listen for commands from the interrogator

— Interrogator: transmit a Collection with Universal Data Block command to the tags.

— Tag: If in the ready state, receives and decodes the Collection with Universal Data Block command,

— Tag: reply with a response packet that includes it's tag identification and the first portion of the requested UDB type.

— Interrogator: for each tag from which the interrogator received a valid response message, the interrogator may send point-to-point Read Universal Data Block commands to retrieve any remaining UDB from the tag, and then the interrogator shall send a point-to-point Sleep command to the tag.

— Tag: will respond to any point-to-point Read Universal Data Block commands, which may be received from the interrogator.

— Tag: on receiving a Sleep Command, shall leave the ready state and shall not respond to further commands from the interrogator until after the tag receives a Wake Up Signal.

7.4.4.6 Multi-packet UDB Collection

The following section provides a simple example of a multi-packet UDB Collection. Typically an interrogator will initiate the process by transmitting a broadcast Collect with UDB command. Tags that receive the Collect command will reply with a response packet that includes their tag identification and the first portion of the requested UDB type. The interrogator can then retrieve the remaining UDB data by sending a point-to-point Read UDB command to each tag identified in the first phase.

1. Figure 34 shows the interrogator's Collection with UDB command packet. This is a broadcast packet and all tags that receive the packet will participate in the collection process.

Extended protocol ID	Options	Payload Length	Command Code	Windows size	Max Packet Length	UDB Type Code
0x01	1 byte	1 byte	0x1F	2 bytes	1 byte	1 byte

Figure 34 — Interrogator Collect with UDB broadcast for the initial collection of the tags

2. Figure 35 illustrates a tag response packet to the broadcast Collection with UDB command (command code 0x1F). The reply is in the format of a broadcast response packet

Extended protocol ID	Tag Status	Payload Length	Command Code	UDB Type Code	Total UDB Length	Requested Offset	Universal Data Block
0x01	2 bytes	1 byte	0x1F	1 byte	2 bytes	2 bytes	N bytes

Figure 35 — Tag Read UDB response to interrogator's Collection with UDB command

3. Figure 36 shows the interrogator's Read Universal Data Block command directed to a tag identified during the previous collection. The point-to-point command is directed to a specific tag using the tag identification value discovered in the previous collection. Note that the Offset into UDB field will contain a value equal to the number of bytes of UDB data returned in the Tag's collection response (N in Figure 35 above).

Extended protocol ID	Options	Payload Length	Command Code	UDB Type Code	Offset into UDB	Max Packet Length
0x01	1 byte	1 byte	0x70	1 byte	2 byte	1 byte

Figure 36 — Interrogator's Read UDB request to a specific tag for the remainder of the UDB data

4. Figure 37 shows the tag reply in response to the interrogator's Read UDB command. The reply contains the second part of the UDB message using the point-to-point (directed) packet format.

Extended protocol ID	Tag Status	Payload Length	Command Code	UDB Type Code	Total UDB Length	Requested Offset	Universal Data Block
0x01	2 bytes	1 byte	0x70	1 byte	2 bytes	2 bytes	N bytes

Figure 37 — Tag response to interrogator's request for additional information with a point–to-point Read UDB command

7.4.5 Extended Services

This clause describes the how application services are defined on an Extended Mode device.

7.4.5.1 Method for Extending ISO/IEC 18000-7 Extended Mode Application Functionality

To extend functionality such as sensors, or other features into ISO/IEC 18000-7 Extended Mode Application the data link shall include a simple message transport capability. This message transport is provided by a set of **Extended Service** commands that encapsulate the application's message for transport across the physical layer to enable infrastructure devices (interrogators) to control and access services (applications) resident in the tags. These extended service applications can share resources within the tag, and also connect to external devices using secondary communications (e.g. GPS, wireless sensors, etc.). Additionally, UDB elements are added to enable discovery and support of these extended services. Thus, with the **Extended Service** option the functionality of the tag can be extended without further additional changes to the standard.

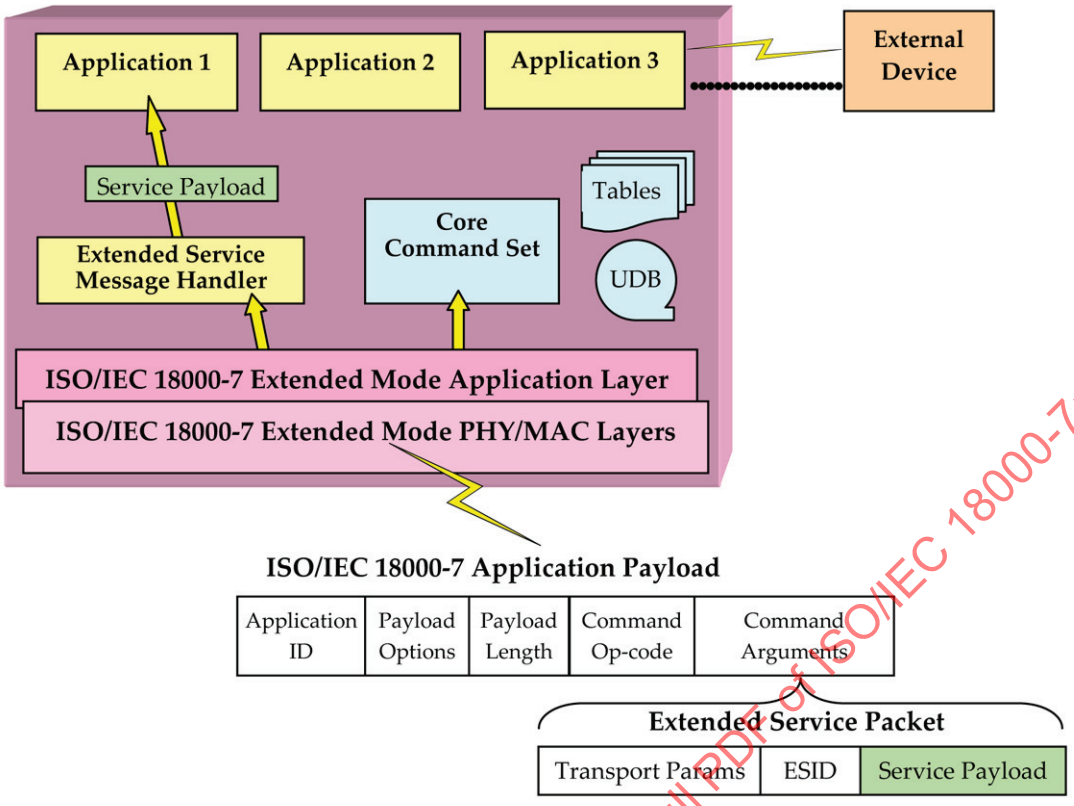


Figure 38 — Extended Service Support

7.4.5.1.1 Overview of the Extended Service option

The Extended Service option includes three commands (*Extended Service Tx*, *Extended Service Rx*, and *Extended Service TxRx*) that provide transport of messages, and two new UDB elements to allow extended services to notify clients of their presence and identity (the E*xtended Services List Element*), and to provide alarm information (the *Alarm Summary Element*).

Within the tag, Extended Service applications are identified using an Extended Service ID (ESID). The ESID is a unique value in the range of 0x80 to 0x8F (value of 0x00 through 0x7F are reserved) that is assigned by the tag for each application it supports. The ESID for an application is determined by the tag when the application first becomes available and shall remain constant while the application is available. However, the ESID for an application may or may not be constant for every tag. For example, one tag may assign an ESID value of 0x80 to application 'Foo' whereas another may assign an ESID value of 0x9A to the same application. The application's identity and ESID value can be discovered in the Extended Service List UDB element. The ESID is used in the Extended Service commands to route the messages to the application and to identify the owner of an alarm in the Alarm Summary Element.

Extended Service applications have an input and output mailbox for messages. Messages sent to an application (typically commands) go to the input mailbox which can only be written; messages received from application (typically command responses) go into the output mailbox which can only be read. Each message has an associated ESID and Message ID for routing and linking. If a message initiated requires a response then the response message shall contain the same Message ID value.

The Extended Service Tx and Extended Service Rx commands only transport a message to or from the application's mailbox and reply that the message was delivered or received. They do not wait for the application's response. This decouples the timing and processing between the transport and the application and enables asynchronous processing by the application. Additionally, the Extended Service TxRx command is provided to support a round trip synchronous transfer in situations where an application's response is immediate and asynchronous processing would be less efficient.

7.4.5.1.2 Changes and Additions to ISO/IEC 18000-7 Extended Mode Application Support

The following are the changes and additions to ISO/IEC 18000-7 Extended Mode Application Support to support the proposed Extended Service functionality. Where applicable, specific sections, tables, figures, etc. in the standard are noted.

7.4.5.1.3 Extended Service Tx

This command transports a message packet to a specified Extended Service application's input mailbox. Multiple consecutive invocations of this command can be used to transmit a sequence of message packets that form a complete message. After the last message packet of a sequence has been transmitted asynchronous processing of that message is initiated. This command may also be used with the Extended Service TxRx command to initiate synchronous messages.

Table 317 — Extended Service Tx command format

Command Code	Sub Command	Sequence ID	ESID	Message ID	Message Size	Offset	Data Size	Data
0x27	0x00	1 byte	1 byte	1 byte	2 bytes	2 bytes	1 byte	N bytes

- **Sequence ID:** Identifies this message packet in a sequence of message packets to be written. For a sequence of N packets, the Sequence ID is N-1 for the first packet, N-2 for the second packet, and so forth, down to 0 (zero) for the Nth packet. The maximum Message Size and the number of packets per message are implementation dependent.
- **ESID:** The identifier of the application to receive this message packet.
- **Message ID:** The unique identifier of the message. If a complete message requires multiple message packets then all message packets shall use the same identifier. The value of 0x00 is reserved. If a response message is associated with this message then it shall contain the same Message ID value.
- **Message Size:** The total number of bytes of the complete message. If this is the second or greater message packet for a previous message then the Message Size must be the same as previous message.
- **Offset:** The offset into the message for where this data is to be written. An offset value of 0x0000 is the first byte of the message.
- **Data Size:** The number of bytes (1…50) of message data in this packet.
- **Data:** The message bytes (application's payload) to be delivered.

When the tag receives an Extended Service Tx message packet it shall write the data into the input mailbox of the ESID at the offset of the message specified by Message ID. The tag shall then respond with a point-to-point response with parameters shown below. If the Sequence ID equals zero (0) then, after sending the response, the tag shall notify the application that a complete message has been received. Note the application's response, if any, is not returned with this command. Refer to the Extended Service Rx command to obtain the application's response.

Table 318 — Extended Service Tx response format

Command Code	Sequence ID	ESID	Message ID	Total Message Bytes
0x27	1 byte	1 byte	1 byte	2 bytes

- **Sequence ID:** The sequence ID of the completed command packet.
- **ESID:** The identifier of the application receiving this message packet.
- **Message ID:** The unique identifier of the message.

– **Total Message Written:** The total number of bytes of the message that has been written.

Table 319 — Extended Service Tx command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	• Sequence ID is not the same value or one less than the value of the previous Extended Service Tx command with the same Message ID, or… • ESID is invalid or no application is associated with it, or… • Message ID is zero or does not match the previous invocation of this command where the Sequence ID was not zero, or… • Message Size is zero or does not match a previous invocation of this command with the same Message ID which is in progress, or… • The Offset into Message ID is greater than the total length of the message, or… • Data Size is zero, or the length of the Data parameter does not agree, or… • The wrong number of parameters bytes was supplied.
0x0A	Operation Failed	• Write operation failed or the application's input mailbox is full

7.4.5.1.4 Extended Service Rx

The Extended Service Rx command transports a message packet from a specified Extended Service application's output mailbox. This command may be used to transport a sequence of message packets that form a complete message. The tag responds to each command with an acknowledgment that the message packet was sent.

Table 320 — Extended Service Rx command format

Command Code	Sub Command	ESID	Message ID	Max Packet Size	Offset
0x27	0x01	1 byte	1 byte	1 byte	2 bytes

– **ESID:** The identifier of the application requesting data from.

– **Message ID:** The unique identifier of the message that corresponds to an input message received previously. The value of 0x00 indicates to return the first available message.

– **Max Packet Size:** This value of 22 to 255 specifies the maximum number of bytes a tag can send and use as the Packet Length field of its response. Tags may use a different response Packet Length providing the value does not exceed the value of Max Packet Length. The value 22 is the minimum tag response packet which includes 15 bytes for response packet overhead, 1 byte for the ESID value, 1 byte for the Message ID value, 2 bytes for the Message Size value, 2 bytes for the Requested Offset value, and 1 byte of message data.

– **Offset:** The offset into the message from which to begin reading.

When the tag receives an Extended Service Rx response packet it shall respond with a point-to-point response message with command code, parameters, and data as shown. If a complete message is not available in the application's output mailbox then the Message Size shall be 0x0000 and no Data bytes are returned. Tags may select a smaller packet size than specified by Max Packet Length but may not exceed that value. After successfully receiving the initial portion of the message, the Interrogator may use subsequent invocations of this command to read the remaining message bytes by advancing the Offset value to the next unread data byte position. The interrogator does not have to read the entire message; however, the message may not remain upon receipt of the next input message.

Table 321 — Extended Service Rx response format

Command Code	ESID	Message ID	Message Size	Requested Offset	Data
0x27	1 byte	1 byte	2 bytes	2 bytes	N bytes

- **ESID:** The identifier of the application providing message packet.
- **Message Size:** The total number of bytes in mailbox message.
- **Message ID:** The unique identifier of the message. A value of 0x00 indicates this message is not associated with a previous input message (i.e. it may be an alert message).
- **Requested Offset:** The offset into the mailbox of the data returned
- **Data:** The returned output mailbox data

Table 322 — Extended Service Tx command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	• ESID is invalid or no application is associated with it, or... • Max Packet Length is less than 22, or... • The Offset into Message ID is greater than the total length of the message, or... • The wrong number of parameters bytes was supplied.

7.4.5.1.5 Extended Service TxRx

This command is used when a message is transmitted to an application that requires a synchronous response. This command may be used either to transmit a single packet message or in conjunction with an Extended Service Tx command to transmit the last message packet of a multi-packet message.

Table 323 — Extended Service TxRx command format

Command Code	Sub Command	ESID	Message ID	Message Size	Offset
0x27	0x02	1 byte	1 byte	2 bytes	2 bytes
Data Size	Data	Max Packet Size	Response Time		
1 byte	N bytes	1 byte	1 byte		

- **ESID:** The identifier of the application to receive this message packet.
- **Message Size:** The total number of bytes of the complete message. If this is the second or greater message packet for a previous message then the Message Size must be the same as previous message.
- **Message ID:** The unique identifier of the message. If this message packet is the last of a sequence of message packets forming a complete message then this value shall be the same as used with previous commands. The value of 0x00 is reserved. If a response message is associated with this message then it shall contain the same Message ID value.
- **Offset:** The offset into the message for where this data is to be written. An offset value of 0x0000 is the first byte of the message.
- **Data Size:** The number of bytes (1...50) of message data in this packet.
- **Data:** The message bytes to be written.

- **Max Packet Size:** This value of 20 to 255 specifies the maximum number of bytes a tag can send and use as the Packet Length field of its response. Tags may use a different response Packet Length providing the value does not exceed the value of Max Packet Length. The value 20 is the minimum tag response packet which includes 15 bytes for response packet overhead, 1 byte for the ESID value, 1 byte for the Message ID value, 2 bytes for the Message Size value, and 1 byte of message data.
- **Response Time:** The maximum time to wait in 10 millisecond increments for the application's completed response message before returning a response.

When the tag receives an Extended Service TxRx command it shall write the data into the input mailbox of the ESID at the offset of the message specified by Message ID. If, after writing the data, all bytes of the message have been received the tag shall notify the application a complete message is ready for processing. The tag shall then monitor the output mailbox for a complete response message until the time specified by Response Time has elapsed.

When the application's response is available the tag shall transmit the response with the command code, parameters, and data as shown. Tags may select a smaller packet size than specified by Max Packet Length but may not exceed that value. After successfully receiving the initial portion of the message, the Interrogator may use subsequent invocations of this command to read the remaining message bytes by advancing the Offset value to the next unread data byte position. The interrogator does not have to read the entire message; however, the message may not remain upon receipt of the next input message.

Table 324 — Extended Service TxRx response format

Command Code	ESID	Message ID	Message Size	Data
0x27	1 byte	1 byte	2 bytes	N bytes

- **ESID:** The identifier of the application providing this message packet.
- **Message ID:** The unique identifier of the message.
- **Message Size:** The total number of bytes of the message.
- **Data:** The message bytes (application's payload) received.

Table 325 — Extended Service RxTx command errors

Error Code	Error Name	Reason
0x02	Invalid Command Parameter	• ESID is invalid or no application is associated with it, or... • Message ID is zero or does not match the previous invocation of an Extended Service Tx command where the Sequence ID was not zero, or... • Message Size is zero or does not match a previous invocation of Extended Service Tx with the same Message ID which is in progress, or... • Max Packet Length is less than 20, or... • The wrong number of parameters bytes was supplied.
0x0A	Operation Failed	• Write operation failed or the application's input mailbox is full

7.4.5.1.6 UDB Elements

The following ISO/IEC 18000-7 Extended Mode Application Support tables have been updated to indicate the new UDB Elements associated with the **Extended Service** capability: