INTERNATIONAL STANDARD

ISO/IEC 18000-7

Third edition
2009-08-01

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 pour la gestion d'objets —

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

Reference number
ISO/IEC 18000-7:2009(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.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

ISO/IEC 18000-7 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 31, *Automatic identification and data capture techniques*.

This third edition cancels and replaces the second edition (ISO/IEC 18000-7:2008), which has been technically revised.

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*

— *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.

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 a patent concerning radio frequency identification technology given in 6.2. ISO and IEC take no position concerning the evidence, validity and scope of this patent right.

The holder of this patent right has assured ISO and IEC that he is willing to negotiate licenses under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with ISO and IEC. Information may be obtained from the following.

Patent number	Patent title	Patent holder	Contact	Affected subclause
US 5640151	Communication system for communicating with tags	Savi Technology	Hurst Arthur, VP, General Counsel, Savi Technology, Inc., 351 East Evelyn Ave., Mountain View, CA 94041, USA	6.2.6
US 5686902	Communication system for communicating with tags	Savi Technology	Hurst Arthur, VP, General Counsel, Savi Technology, Inc., 351 East Evelyn Ave., Mountain View, CA 94041, USA	6.2.6
US 11/432684	Method and apparatus for efficient data transmission from a tag	Savi Technology	Hurst Arthur, VP, General Counsel, Savi Technology, Inc., 351 East Evelyn Ave., Mountain View, CA 94041, USA	

EP 0467036	Method and apparatus for radio identification and tracking	Savi Technology	Hurst Arthur, VP, General Counsel, Savi Technology, Inc., 351 East Evelyn Ave., Mountain View, CA 94041, USA	6.2.6
US 6002344	System and method for electronic inventory	Symbol Technologies	Aaron B. Bernstein, VP, Deputy General Counsel Intellectual Property, 1 Motorola Plaza, MS A6, Holtsville, NY 11561, USA	6.2
CA 2,310,623	System and method for electronic inventory	Symbol Technologies	Aaron B. Bernstein, VP, Deputy General Counsel Intellectual Property, 1 Motorola Plaza, MS A6, Holtsville, NY 11561, USA	6.2
CN 98812462.9	System and method for electronic inventory	Symbol Technologies	Aaron B. Bernstein, VP, Deputy General Counsel Intellectual Property, 1 Motorola Plaza, MS A6, Holtsville, NY 11561, USA	6.2
DE 98960332.9	System and method for electronic inventory	Symbol Technologies	Aaron B. Bernstein, VP, Deputy General Counsel Intellectual Property, 1 Motorola Plaza, MS A6, Holtsville, NY 11561, USA	6.2
EP 98960332.9	System and method for electronic inventory	Symbol Technologies	Aaron B. Bernstein, VP, Deputy General Counsel Intellectual Property, 1 Motorola Plaza, MS A6, Holtsville, NY 11561, USA	6.2
FR 98960332.9	System and method for electronic inventory	Symbol Technologies	Aaron B. Bernstein, VP, Deputy General Counsel Intellectual Property, 1 Motorola Plaza, MS A6, Holtsville, NY 11561, USA	6.2
GB 98960332.9	System and method for electronic inventory	Symbol Technologies	Aaron B. Bernstein, VP, Deputy General Counsel Intellectual Property, 1 Motorola Plaza, MS A6, Holtsville, NY 11561, USA	6.2
HK 01101416.3	System and method for electronic inventory	Symbol Technologies	Aaron B. Bernstein, VP, Deputy General Counsel Intellectual Property, 1 Motorola Plaza, MS A6, Holtsville, NY 11561, USA	6.2
IL 136.220	System and method for electronic inventory	Symbol Technologies	Aaron B. Bernstein, VP, Deputy General Counsel Intellectual Property, 1 Motorola Plaza, MS A6, Holtsville, NY 11561, USA	6.2
IT 98960332.9	System and method for electronic inventory	Symbol Technologies	Aaron B. Bernstein, VP, Deputy General Counsel Intellectual Property, 1 Motorola Plaza, MS A6, Holtsville, NY 11561, USA	6.2
JP 2000-521687	System and method for electronic inventory	Symbol Technologies	Aaron B. Bernstein, VP, Deputy General Counsel Intellectual Property, 1 Motorola Plaza, MS A6, Holtsville, NY 11561, USA	6.2
US 7,035,818	System and method for electronic inventory	Symbol Technologies	Aaron B. Bernstein, VP, Deputy General Counsel Intellectual Property, 1 Motorola Plaza, MS A6, Holtsville, NY 11561, USA	6.2
US 10/725,010	System and method for electronic inventory	Symbol Technologies	Aaron B. Bernstein, VP, Deputy General Counsel Intellectual Property, 1 Motorola Plaza, MS A6, Holtsville, NY 11561, USA	6.2
US 10,932,279	System and method for electronic inventory	Symbol Technologies	Aaron B. Bernstein, VP, Deputy General Counsel Intellectual Property, 1 Motorola Plaza, MS A6, Holtsville, NY 11561, USA	6.2
US 6,470,045	Communication protocol between a transceiver unit and transponders or transceiver associated with said unit	EM Microelectronic Marin SA	G. Meusburger, IP Manager, Rue des Sors, CH-2074, Marin, Switzerland	
JP 10-256493	Communication protocol between a transceiver unit and transponders or transceiver associated with said unit	EM Microelectronic Marin SA	G. Meusburger, IP Manager, Rue des Sors, CH-2074, Marin, Switzerland	

EP 0 902 546 Appl. No. 97115772.2	Communication protocol between a transceiver unit and transponders or transceiver associated with said unit	EM Microelectronic Marin SA	G. Meusburger, IP Manager, Rue des Sors, CH-2074, Marin, Switzerland	
US 6,784,787 Granted EP 1 031 046 Granted EP 1 291 671 Granted EP Appl 05 017 862.3 Pending	Identification systems	Zebra Technologies	Eric McAlpine, IP Counsel, Legal Department, 333 Corporate Woods Parkway, Vernon Hills, IL 60061-3109, USA	
US 6,480,143 Granted EP 1 001 366 Granted	Electronic identification systems	Zebra Technologies	Eric McAlpine, IP Counsel, Legal Department, 333 Corporate Woods Parkway, Vernon Hills, IL 60061-3109, USA	
US 5,680,459 Granted	Passive transponder	Zebra Technologies	Eric McAlpine, IP Counsel, Legal Department, 333 Corporate Woods Parkway, Vernon Hills, IL 60061-3109, USA	
US 6,198,381 Granted JP 10-272945 Pending	Delayed reset mode model for electronic identification system	Zebra Technologies	Eric McAlpine, IP Counsel, Legal Department, 333 Corporate Woods Parkway, Vernon Hills, IL 60061-3109, USA	
US 5,537,105 Granted US 5,966,083 Granted US 5,995,017 Granted	Electronic identification systems	Zebra Technologies	Eric McAlpine, IP Counsel, Legal Department, 333 Corporate Woods Parkway, Vernon Hills, IL 60061-3109, USA	
US 7375637 Granted	Methods and apparatus for reducing power consumption of an active transponder	University of Pittsburgh	Marc S. Malandro, Ph.D., CLP, University of Pittsburgh, 200 Gardner Steel Conference Center, Thackeray & O'Hara Streets, Pittsburgh, PA 15260, USA	
US 11/678296 Pending	Methods and apparatus for switching a transponder to an active state, and asset management systems employing same	University of Pittsburgh	Marc S. Malandro, Ph.D., CLP, University of Pittsburgh, 200 Gardner Steel Conference Center, Thackeray & O'Hara Streets, Pittsburgh, PA 15260, USA	
US 61/099977 Pending	System and method for real time asset location and tracking	University of Pittsburgh	Marc S. Malandro, Ph.D., CLP, University of Pittsburgh, 200 Gardner Steel Conference Center, Thackeray & O'Hara Streets, Pittsburgh, PA 15260, USA	
US 6563417 Granted US 6917291 Granted US 7053777 Granted	Interrogation, monitoring and data exchange using RFID Tags	Identec Solutions	Stefan Schwiers, CTO, R&D Department, Identec Solutions AG, Millennium Park 2, 6890 Lustenau, Austria	6.3.12

EP 99117640.5-2215 Granted DE 59904147.1-08 Granted GB/FR/CH/NL/AT 99117640.5-2215 Granted	System for monitoring, tracking, and handling of objects	Identec Solutions	Stefan Schwiers, CTO, R&D Department, Identec Solutions AG, Millennium Park 2, 6890 Lustenau, Austria	6.3.12
US 7345576 Granted	Method and apparatus for resolving RFID based object traffic transactions to single container in the presence of a plurality of containers	Identec Solutions	Stefan Schwiers, CTO, R&D Department, Identec Solutions AG, Millennium Park 2, 6890 Lustenau, Austria	6.3.12
		Impinj	Chris Diorio, CTO, 701 N. 34th Street, Suite 300, Seattle, WA 98103, USA	
		Intermec	Phyllis T. Turner-Brim, Esq., Legal Department, Intermec IP Corporation, 6001 36th Ave. W, Everett, WA 98203, USA	

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 or IEC shall not be held responsible for identifying any or all such patent rights.

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 referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest published 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*

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, the symbols and abbreviated terms given in ISO/IEC 19762-1 and ISO/IEC 19762-3 apply.

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 readers and respond accordingly. When the tag is asleep, it will ignore all commands.

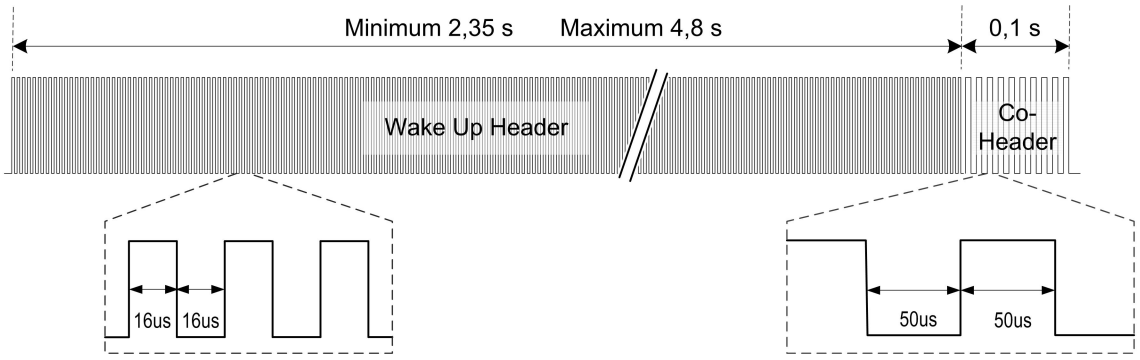


Figure 1 — Wake Up Signal

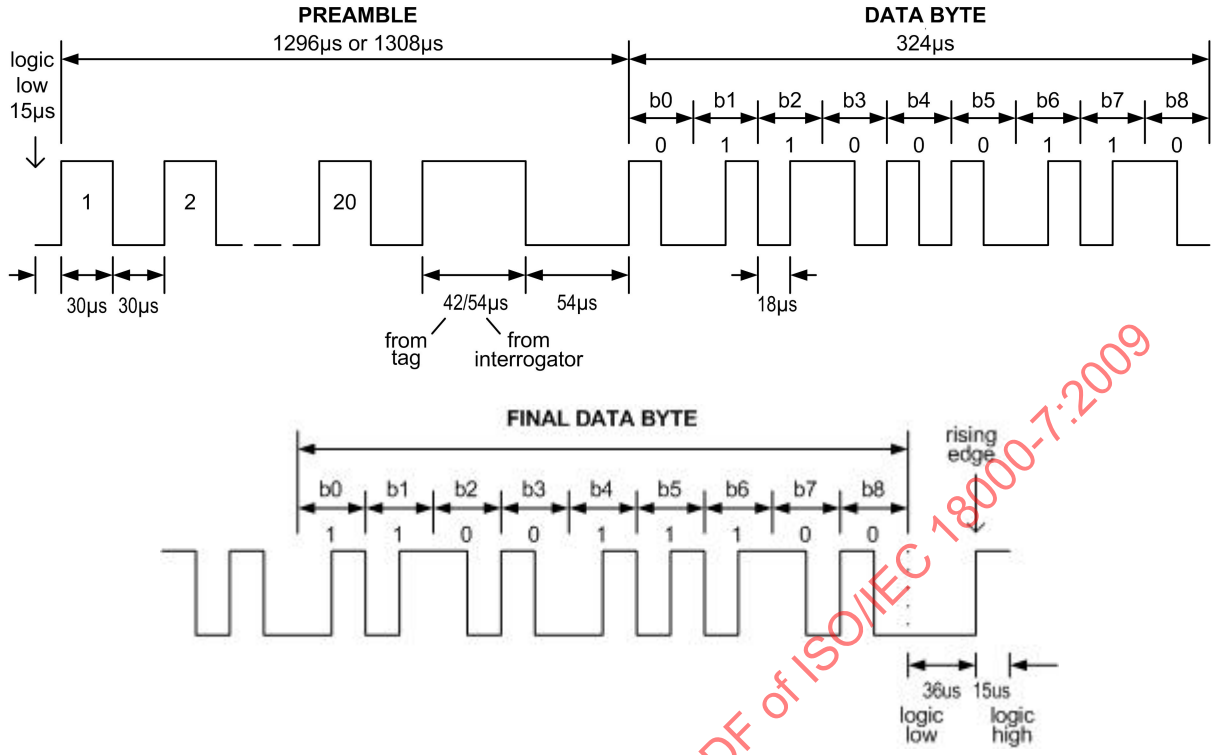
Once awoken, the tag shall stay awake for a minimum of 30 seconds after receipt of the last well-formed message packet consisting of a valid Protocol ID, command code, and CRC values, unless the interrogator otherwise commands the tag to sleep. If no well-formed command message is received within the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

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[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

1) Bit 2 of the “packet options 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/IEC15963 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.

autoid.org is the ISO/IEC 18000-7 Registration Authority.

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
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 checksum 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.

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	Acknowledgement 1 = 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 57,3 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 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 (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
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.

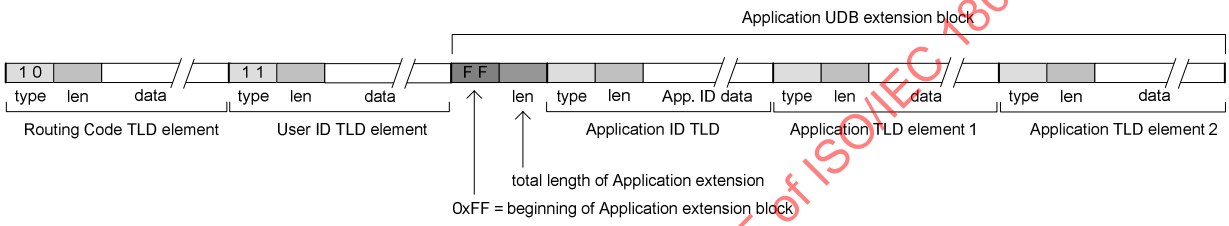


Figure 3 — Example Universal Data Block - UDB Type 0x00

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.

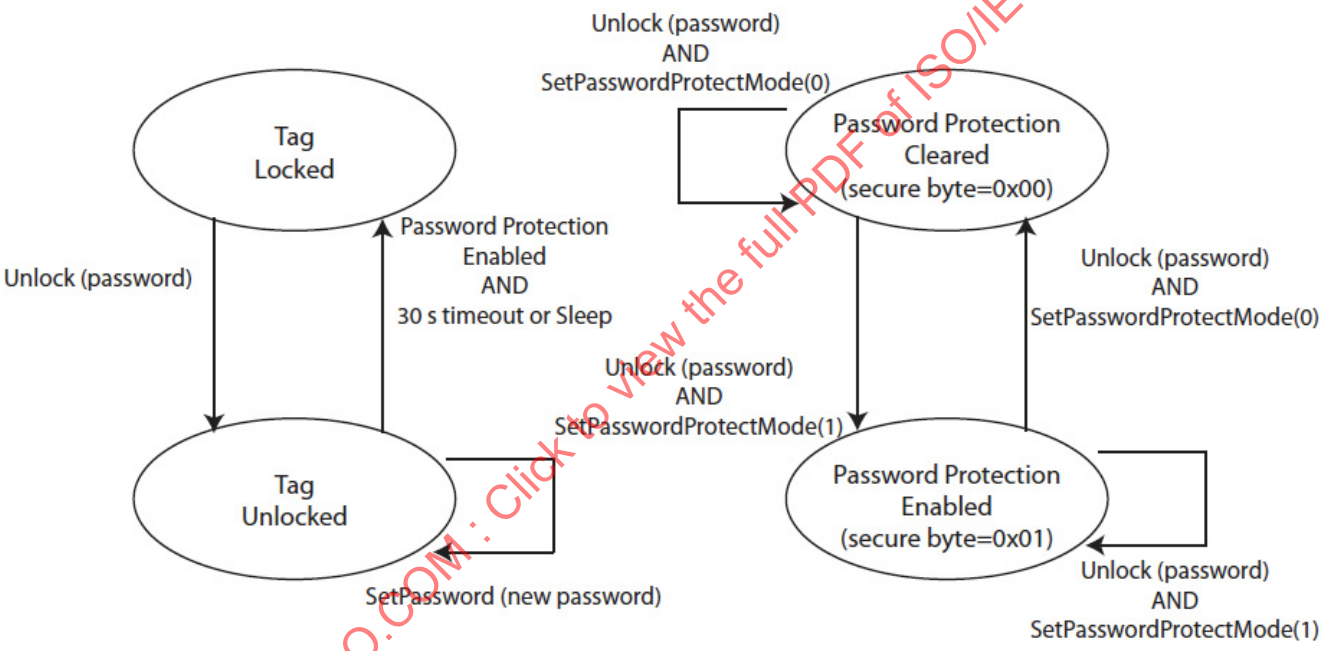


Figure 4 - Tag security state machines

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
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

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

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 15459-1. 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:

NEN - Nederlands Normalisatie-instituut - Registration Authority of ISO/IEC 15459
Postbus 5059
2600 GB Delft
THE NETHERLANDS
Fax: + 31 15 26 90 242
E-mail: RA-ISO15459@nen.nl

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	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

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	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

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
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

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 was attempted with password protection engaged and tag in the locked state
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 reader 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 - 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. 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 two 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
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 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

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 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

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 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

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 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

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, and Table Update Fields, Table Delete Record, 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.