INTERNATIONAL STANDARD

ISO/IEC 18000-63

Second edition
2015-10-15

Information technology — Radio frequency identification for item management —

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

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

Partie 63: Paramètres de communications d'une interface radio entre 860 MHz et 960 MHz, Type C

Reference number
ISO/IEC 18000-63:2015(E)

ISO copyright office
Ch. de Blandonnet 8 • CP 401
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

Contents

Page

Foreword

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

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

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

Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

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

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

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

This second edition cancels and replaces the first edition (ISO/IEC 18000-63:2013), 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 General*

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

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

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

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

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

Introduction

This part of ISO/IEC 18000 defines the physical and logical requirements for a passive-backscatter, Interrogator-talks-first (ITF), radio-frequency identification (RFID) system operating in the 860 MHz – 960 MHz frequency range. The system comprises Interrogators, also known as Readers, and Tags, also known as Labels or Transponders.

An Interrogator transmits information to a Tag by modulating an RF signal in the 860 MHz – 960 MHz frequency range. The Tag receives both information and operating energy from this RF signal. Tags are passive, meaning that they receive all of their operating energy from the Interrogator's RF signal.

An Interrogator receives information from a Tag by transmitting a continuous-wave (CW) RF signal to the Tag; the Tag responds by modulating the reflection coefficient of its antenna, thereby backscattering an information signal to the Interrogator. The system is ITF, meaning that a Tag modulates its antenna reflection coefficient with an information signal only after being directed to do so by an Interrogator.

Interrogators and Tags are not required to talk simultaneously; rather, communications are half-duplex, meaning that Interrogators talk and Tags listen, or vice versa.

The described backscatter radio frequency identification (RFID) system that supports the following system capabilities:

— identification and communication with multiple tags in the field;

— selection of a subgroup of tags for identification or with which to communicate;

— reading from and writing to or rewriting data many times to individual tags;

— user-controlled permanently lockable memory;

— data integrity protection;

— Interrogator-to-tag communications link with error detection;

— tag-to-Interrogator communications link with error detection;

— support for both passive back-scatter tags with or without batteries.

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

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

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

Information on the declared patents may be obtained from:

Contact details
Patent Holder: Legal Name Atmel Automotive GmbH **Contact for license application:** Name & Department Leo Merken, Legal Department, ATMEL Corporation Address 2325 Orchard Parkway Address San Jose, CA 95131 USA Tel. +1 408 436 4251 Fax +1 408 487 2615 E-mail leo.merken@atmel.com URL (optional)
Patent Holder: Legal Name CISC Semiconductor Design+Consulting GmbH **Contact for license application:** Name & Department Markus Pistauer, CEO Address Lakeside B07 Address 9020 Klagenfurt, Austria Tel. +43(463) 508 808 Fax +43(463) 508 808-18 E-mail m.pistauer@cisc.at URL (optional) www.cisc.at
Patent holder: ETRI (Electronics Telecommunication Reseach Institute) **Contact for license application:** Name & Department: Min-Sheo Choi, Intellectual Property Management Team Address: 138 Gajeongno, Yuseong-gu Address: Daejeon, 305-700, Korea Tel. +82-42-860-0756 Fax +82-42-860-3831 E-mail choims@etri.re.kr URL (optional) www.etri.re.kr

Contact details
Patent Holder: Legal Name Impinj, Inc. **Contact for license application:** Name & Department Chris Diorio, CTO Address 701 N. 34th Street, Suite 300 Address Seattle, WA 98103, USA Tel. +1.206 834 1115 Fax +1.206 517.5262 E-mail diorio@impinj.com URL (optional) www.impinj.com
Patent Holder: Legal Name: Magellan Technology Pty. Limited **Contact for license application:** Name & Department: Ms Jean Angus Address: 65 Johnston St Address: Annandale, NSW 2038, Australia Tel. +61 2 9562 9800 Fax +61 2 9518 7620 E-mail: license@magellan-technology.com URL (optional): www.magellan-technology.com
Patent Holder: Legal Name NXP B.V. **Contact for license application:** Name & Department Aaron Waxler – IP Licensing & Claims Address 411 East Plumeria, Address San Jose, CA 95134-1924, USA Tel. +1 914 860-4296 Fax E-mail Aaron.Waxler@nxp.com URL (optional)

Contact details
Patent Holder: Legal Name SATO VICINITY Pty. Limited **Contact for license application:** Name & Department Mr. Hiromasa Konishi, Managing Director Address 8 Guihen Street, Annandale, NSW 2038, Australia Address Tel. +61 295 629 800 Fax +61 295 187 620 E-mail hiromasa.konishi@sato-glo bal . com URL (optional) www.satovicinity.com
Patent Holder: Legal Name TAGSYS SAS **Contact for license application:** Name & Department Mr. Alain Fanet President Address 785 Voie Antiope, TI Athélia 3 Address F-13600 La Ciotat Tel. +33 332188900 Fax +33 332188900 E-mail alain.fanet@tagsysrfid.com URL (optional) www.tagsysrfid.com
Patent Holder: Legal Name Univertsity of Pittsburgh - Of the Commonwealth of Pennsylvania **Contact for license application:** Name & Department Marc S. Malandro, PhD, CLP, RTIP Address University of Pittsburgh, 200 Gardner Steel Conference Center Address Thackeray & O'Hara Streets, Pittsburgh, PA 15260 Tel. 412-624-8787 Fax 412-648-2259 E-mail mmalandro@innovation.pitt.edu URL (optional)

Contact details	
Patent Holder:	
Legal Name	Zebra Technologies Corporation
Contact for license application:	
Name & Department	Glenn Frankenberger, Sr. IP Counsel, Legal Department
Address	One Motorola Plaza
Address	Holtsville, NY 11742
Tel.	631-738-5570
Fax	631-738-4110
E-mail	glenn.frankenberger@zebra.com
URL (optional)	

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

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

Information technology — Radio frequency identification for item management —

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

1 Scope

This part of ISO/IEC 18000 defines the air interface for radio frequency identification (RFID) devices operating in the 860 MHz to 960 MHz Industrial, Scientific, and Medical (ISM) band used in item management applications. It provides a common technical specification for RFID devices that can be used by ISO 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. It defines the forward and return link parameters for technical attributes including, but not limited to, operating frequency, operating channel accuracy, occupied channel bandwidth, maximum effective isotropic radiated power (EIRP), 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. It further defines the communications protocol used in the air interface.

This part of ISO/IEC 18000 specifies the physical and logical requirements for a passive-backscatter, Interrogator-Talks-First (ITF) systems. The system comprises Interrogators, also known as readers, and tags, also known as labels. An Interrogator receives information from a tag by transmitting a continuous-wave (CW) RF signal to the tag; the tag responds by modulating the reflection coefficient of its antenna, thereby backscattering an information signal to the Interrogator. The system is ITF, meaning that a tag modulates its antenna reflection coefficient with an information signal only after being directed to do so by an Interrogator.

In detail, this part of ISO/IEC 18000 contains Type C.

Type C uses PIE in the forward link and a random slotted collision-arbitration algorithm.

This part of ISO/IEC 18000 specifies

— physical interactions (the signalling layer of the communication link) between Interrogators and tags,

— logical operating procedures and commands between Interrogators and Tags.

— the collision arbitration scheme used to identify a specific tag in a multiple-tag environment.

— optional security commands that allow the use of crypto suites of ISO/IEC 29167.

2 Conformance

2.1 Claiming conformance

A device shall not claim conformance with this protocol unless the device complies with

— all clauses in this protocol (except those marked as optional), and

— the conformance document associated with this protocol, and,

— all local radio regulations.

Relevant conformance test methods are provided in ISO/IEC 18047-6.

Conformance can also require a license from the owner of any intellectual property utilized by said device.

2.2 General conformance requirements

2.2.1 Interrogators

To conform to this part of ISO/IEC 18000, an Interrogator shall:

— Meet the requirements of this part of ISO/IEC 18000,

— Implement the mandatory commands defined in this part of ISO/IEC 18000,

— Modulate/transmit and receive/demodulate a sufficient set of the electrical signals defined in the signaling layer of this protocol to communicate with conformant Tags, and

— Conform to the applicable local radio regulations.

To conform to this part of ISO/IEC 18000, an Interrogator may:

— Implement any subset of the optional commands defined in this part of ISO/IEC 18000, and

— Implement any proprietary and/or custom commands in conformance with this part of ISO/IEC 18000.

To conform to this part of ISO/IEC 18000, an Interrogator shall not:

— implement any command that conflicts with this part of ISO/IEC 18000 or any of the parts 61, 62 and 64, or

— Require using an optional, proprietary, or custom command to meet the requirements of this protocol.

2.2.2 Tags

To conform to this part of ISO/IEC 18000, a Tag shall:

— Meet the requirements of this part of ISO/IEC 18000,

— Implement the mandatory commands defined in this part of ISO/IEC 18000,

— Modulate a backscatter signal only after receiving the requisite command from an Interrogator, and

— Conform to local radio regulations.

To conform to this protocol, a Tag may:

— Implement any subset of the optional commands defined in this part of ISO/IEC 18000, and

— Implement any proprietary and/or custom commands as defined in 2.3.3 and 2.3.4, respectively.

To conform to this part of ISO/IEC 18000, a Tag shall not:

— Implement any command that conflicts with this part of ISO/IEC 18000 or any of the parts 61, 62 and 64,

— Require using an optional, proprietary, or custom command to meet the requirements of this protocol, or

— Modulate a backscatter signal unless commanded to do so by an Interrogator using the signaling layer defined in this part of ISO/IEC 18000.

2.3 Command structure and extensibility

This part of ISO/IEC 18000 allows four command types: (1) mandatory, (2) optional, (3) proprietary, and (4) custom. Subclause 6.3.2.12 and Table 6.28 define the structure of the command codes used by Interrogators and Tags for each of the four types, as well as the availability of future extensions. All commands defined by this protocol are either mandatory or optional. Proprietary or custom commands are manufacturer-defined.

2.3.1 Mandatory commands

Conforming Tags shall support all mandatory commands. Conforming Interrogators shall support all mandatory commands.

2.3.2 Optional commands

Conforming Tags may or may not support optional commands. Conforming Interrogators may or may not support optional commands. If a Tag or an Interrogator implements an optional command then it shall implement it in the manner specified in this protocol.

2.3.3 Proprietary commands

Proprietary commands may be enabled in conformance with this protocol, but are not specified herein. All proprietary commands shall be capable of being permanently disabled. Proprietary commands are intended for manufacturing purposes and shall not be used in field-deployed RFID systems.

2.3.4 Custom commands

Custom commands may be enabled in conformance with this protocol, but are not specified herein. An Interrogator shall issue a custom command only after (1) singulating a Tag, and (2) reading (or having prior knowledge of) the Tag manufacturer's identification in the Tag's TID memory. An Interrogator shall use a custom command only in accordance with the specifications of the Tag manufacturer identified in the TID. A custom command shall not solely duplicate the functionality of any mandatory or optional command defined in this protocol by a different method.

2.4 Reserved for Future Use (RFU)

This part of ISO/IEC 18000 denotes some Tag memory addresses, Interrogator command codes, and bit fields within Interrogator commands as RFU.

RFU values are reserved for future extensibility. Third parties, including but not limited to solution providers and end users, shall not use these RFU values for proprietary purposes.

2.5 Cryptographic Suite Indicators

A Tag may support one or more cryptographic suites. The *Challenge* and *Authenticate* commands include a CSI field that specifies a single cryptographic suite. CSI is an 8-bit field with bit values defined below.

- Four most-significant bits: Cryptographic suite assigning authority, as follows:
  - 0000_2 – 0011_2: ISO/IEC 29167
  - 0100_2 – 1100_2: RFU
  - 1101_2: Tag manufacturer
  - 1110_2: GS1
  - 1111_2: RFU
- Four least-significant bits: One of 16 cryptographic suites that the assigning authority may assign.

Example: CSI=00000000_2 is the first and CSI=00000001_2 is the second suite that ISO/IEC 29167 may assign.

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 edition of the referenced document (including any amendments) applies.

ISO/IEC 7816-6, *Identification cards — Integrated circuit cards — Part 6: Interindustry data elements for interchange*

ISO/IEC 15961, *Information technology — Radio frequency identification (RFID) for item management — Data protocol: application interface*

ISO/IEC 15962, *Information technology — Radio frequency identification (RFID) for item management — Data protocol: data encoding rules and logical memory functions*

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

ISO/IEC 18000-1, *Information technology — Radio frequency identification for item management — Part 1: Reference architecture and definition of parameters to be standardized*

ISO/IEC 18047-6:2012, *Information technology — Radio frequency identification device conformance test methods — Part 6: Test methods for air interface communications at 860 MHz to 960 MHz*

ISO/IEC 19762 (all parts), *Information technology — Automatic identification and data capture (AIDC) techniques — Harmonized vocabulary*

ISO/IEC 29167-1: *Information technology — Automatic identification and data capture techniques — Part 1: Security services for RFID air interfaces*

GS1 EPCglobal™: *GS1 EPC™ Tag Data Standard*

4 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 19762 (all parts) and the following apply. Terms and definitions specific to this part of ISO/IEC 18000 that supersede any normative references are as follows:

4.1
air interface
complete communication link between an Interrogator and a Tag including the physical layer, collision-arbitration algorithm, command and response structure, and data-coding methodology

4.2
activation
waking up a tag from the **hibernate** state

4.3
asymmetric key pair
private key and its corresponding public key, used in conjunction with an asymmetric cryptographic suite

4.4
authentication
process of determining whether an entity or data is/are who or what, respectively, it claims to be.

Note 1 to entry: The types of entity authentication referred-to in this protocol are Tag authentication, Interrogator authentication, and Tag-Interrogator mutual authentication. For data authentication see authenticated communications.

4.5
authenticated communication
communication in which message integrity is protected

4.6
battery assistance
battery support for radio frequency communication

4.7
battery assisted mode
working mode of battery assisted tags with non-empty battery

4.8
battery saver mode
battery saving functionality based on low power threshold detection with optional duty cycling

4.9
collision arbitration loop
algorithm used to prepare for and handle a dialogue between an Interrogator and a tag

Note 1 to entry: This is also known as collision arbitration.

4.10
command set
set of commands used to inventory and interact with a Tag population

4.11
continuous wave
typically a sinusoid at a given frequency, but more generally any Interrogator waveform suitable for powering a passive Tag without amplitude and/or phase modulation of sufficient magnitude to be interpreted by a Tag as transmitted data

4.12
cover coding
method by which an Interrogator obscures information that it is transmitting to a Tag.

Note 1 to entry: To cover-code data or a password, an Interrogator first requests a random number from the Tag. The Interrogator then performs a bit-wise EXOR of the data or password with this random number, and transmits the cover-coded string to the Tag. The Tag uncovers the data or password by performing a bit-wise EXOR of the received cover-coded string with the original random number.

4.13
crypto superuser
key with an asserted CryptoSuperuser privilege

4.14
data element
low-level, indivisible data construct.

Note 1 to entry: See *file* (4.20) and *record* (4.54).

4.15
dense-Interrogator environment
operating environment within which most or all of the available channels are occupied by active Interrogators

EXAMPLE 25 active Interrogators operating in 25 available channels.

4.16
dense-Interrogator mode
set of Interrogator-to-Tag and Tag-to-Interrogator signaling parameters used in dense-Interrogator environments

4.17
extended Tag identifier
XTID
memory construct that defines a Tag's capabilities and may include a Tag serial number, further specified in the GS1 EPC Tag Data Standard

4.18
extended temperature range
–40 °C to +65 °C

Note 1 to entry: See *nominal temperature range* (4.37).

4.19
file type
8-bit string that specifies a file's designated type

4.20
file
set of one or more records accessed as a unit

Note 1 to entry: See *record* (4.54) and *data element* (4.14).

4.21
file superuser
an access password or key with a 0011_2 **secured**-state file privilege value

4.22
full-duplex communications
a communications channel that carries data in both directions at once

Note 1 to entry: See *half-duplex communications* (4.25).

4.23
full Function Sensors
sensors that can be user configurable and reset, can produce a detailed set of sensor measurement observations, and can be connected to the RFID tag in a variety of ways

4.24
GS1 EPCglobal™ Application
application whose usage denotes an acceptance of GS1 EPCglobal™ standards and policies

Note 1 to entry: See *ISO Application* (4.31.2).

4.25
half-duplex communications
communications channel that carries data in one direction at a time rather than in both directions at once

Note 1 to entry: See full-duplex communications (4.22).

4.26
hibernate
state of energy saving when the device is not required to be used

4.27
insecure communications
communications in which neither message integrity nor message confidentiality are protected

4.28
interrogator authentication
means for a Tag to determine, via cryptographic means, that an Interrogator's identity is as claimed

4.29
inventoried flag
flag that indicates whether a Tag may respond to an Interrogator

Note 1 to entry: Tags maintain a separate inventoried flag for each of four sessions; each flag has symmetric A and B values. Within any given session, Interrogators typically inventory Tags from A to B followed by a re-inventory of Tags from B back to A (or vice versa).

4.30
inventory round
period initiated by a Query command and terminated by either a subsequent Query command (which also starts a new inventory round), a Select command, or a Challenge command

4.31.1
ISO Application
<data structure> application whose MB01 data structure includes a "1" bit in bit 17h and where bits 18h through 1Fh encode an application family identifier as defined in ISO/IEC 15961-3

4.31.2
ISO Application
<usage> application whose usage denotes an acceptance of ISO standards and policies and where in MB01 18h through 1Fh encode an application family identifier as defined in ISO/IEC 15961-3

Note 1 to entry: See *GS1 EPCglobal™ Application* (4.24).

4.32
key
value used to influence the output of a cryptographic algorithm or cipher

4.33
keyID
numerical designator for a single key

4.34
message authentication code
MAC
code, computed over bits in a message, that an Interrogator or a Tag may use to verify the integrity of the message

4.35
multiple-Interrogator environment
operating environment (4.39) within which a modest number of the available channels are occupied by active Interrogators

EXAMPLE 5 active Interrogators operating in 25 available channels.

4.36
mutual authentication
means for a Tag and an Interrogator to each determine, via cryptographic means, that the others' identity is as claimed

4.37
nominal temperature range
–25 °C to +40 °C

Note 1 to entry: See *extended temperature range* (4.18).

4.38
nonremovable Tag
Tag that a consumer cannot physically detach from an item without special equipment or without compromising the item's intended functionality

Note 1 to entry: See *removable Tag* (4.55).

4.39
operating environment
region within which an Interrogator's RF transmissions are attenuated by less than 90dB

Note 1 to entry: In free space, the operating environment is a sphere whose radius is approximately 1000 m, with the Interrogator located at the center. In a building or other enclosure, the size and shape of the operating environment depends on factors such as the material properties and shape of the building, and may be less than 1000 m in certain directions and greater than 1000 m in other directions.

4.40
operating procedure
Collectively, set of functions and commands used by an Interrogator to inventory and interact with Tags

Note 1 to entry: Also known as the Tag-identification layer.

4.41
passive mode
working mode of passive tags or battery assisted tags with battery drained below a manufacturer-specific threshold

4.42
packetCRC
16-bit cyclic-redundancy check (CRC) code that a Tag calculates over its PC, optional XPC word or words, and UII and backscatters during inventory

4.43
packetPC
protocol-control information that a tag with nonzero-valued XI dynamically calculates and backscatters during inventory

4.44
passive Tag (or passive Label)
Tag (or Label) whose transceiver is powered by the RF field

4.45
password
secret value sent by an Interrogator to a Tag to enable restricted Tag operations

Note 1 to entry: Passwords are not keys.

Note 2 to entry: The only passwords defined by this protocol are the kill and access passwords.

4.46
permalock
permalocked
memory location whose lock status is unchangeable (i.e. the memory location is permanently locked or permanently unlocked)

4.47
persistent memory or persistent flag
memory or flag value whose state is maintained during a brief loss of Tag power

4.48
physical layer
data coding and modulation waveforms used in Interrogator-to-tag and tag-to-Interrogator signalling

4.49
private key
undisclosed or non-distributed key in an asymmetric, or public-private key pair, cipher. A private key is typically used for decryption or digital-signature generation. See public key.

4.50
protocol
collectively, physical layer and Tag-identification layer specification

4.51
public key
disclosed or distributed key in an asymmetric, or public-private key pair, cipher.

Note 1 to entry: A public key is typically used for encryption or signature verification.

Note 2 to entry: See private key (4.49).

4.52
Q
parameter that an Interrogator uses to regulate the probability of Tag response

Note 1 to entry: An Interrogator instructs Tags in an inventory round to load a Q-bit random (or pseudo-random) number into their slot counter; the Interrogator may also command Tags to decrement their slot counter. Tags reply when the value in their slot counter (i.e. their slot – see below) is zero. Q is an integer in the range (0,15); the corresponding Tag-response probabilities range from $2^0 = 1$ to $2^{-15} = 0{,}000031$.

4.53
random-slotted collision arbitration
collision-arbitration algorithm where Tags load a random (or pseudo-random) number into a slot counter, decrement this slot counter based on Interrogator commands, and reply to the Interrogator when their slot counter reaches zero

4.54
record
set of one or more data elements accessed as a unit

Note 1 to entry: See *data element* (4.14) and *file* (4.20).

4.55
removable Tag
Tag that a consumer can physically detach from an item without special equipment and without compromising the item's intended functionality

4.56
secure communication
communication in which message confidentiality is protected

4.57
security
degree of protection against threats identified in a security policy. A system is secure if it is protected to the degree specified in the security policy. See security policy.

4.58
security policy
definition, either explicit or implicit, of the threats a system is intended to address

Note 1 to entry: See *security* (4.57).

4.59
session
inventory process comprising an Interrogator and an associated Tag population

Note 1 to entry: An Interrogator chooses one of four sessions and inventories Tags within that session. The Interrogator and associated Tag population operate in one and only one session for the duration of an inventory round (defined above). For each session, Tags maintain a corresponding **inventoried** flag. Sessions allow Tags to keep track of their inventoried status separately for each of four possible time-interleaved inventory processes, using an independent **inventoried** flag for each process.

4.60
session key
temporary key generated by one or both of Tag and Interrogator and typically used for authenticated and/or secure communications

4.61
simple sensors
sensors that are factory programmed and not user configurable, producing a single output observation such as a fail/pass condition or simple measurement of a particular sensor activity

4.62
simple sensor functionality
functionality whereby sensors provide a valid Simple Sensor data address and transmit Simple Sensor data subsequent to the UII as part of the reply to the *ACK* command (Type C)

4.63
single-interrogator environment
operating environment (defined above) within which there is a single active Interrogator at any given time

4.64
singulation
identifying an individual Tag in a multiple-Tag environment

4.65
slot
point, in an inventory round, at which a Tag may respond

Note 1 to entry: Slot is the value output by a Tag's slot counter; Tags reply when their slot (i.e. the value in their slot counter) is zero.

Note 2 to entry: See *Q* (4.52).

4.66
slotted random anticollision
collision-arbitration algorithm where tags load a random (or pseudo-random) number into a slot counter, decrement this slot counter based on Interrogator commands, and reply to the Interrogator when their slot counter reaches zero

4.67
StoredCRC
16-bit cyclic-redundancy check (CRC) computed over the StoredPC and the UII specified by the length (L) bits in the StoredPC, and stored in UII memory

4.68
StoredPC
protocol-control information stored in UII memory

Note 1 to entry: See *PacketPC* (4.43).

4.69
symmetric key
shared key used in conjunction with a symmetric cipher

4.70
tag authentication
means for an Interrogator to determine, via cryptographic means, that a Tag's identity is as claimed

4.71
tag energized
tag in the **ready**, **arbitrate, reply, acknowledged, open,** or **secured** state

4.72
tag-identification layer
collectively, set of functions and commands used by an Interrogator to inventory and interact with Tags (also known as the operating procedure)

4.73
tag not energized
tag not in the **ready**, **arbitrate, reply, acknowledged, open,** or **secured** state

4.74
tari
reference time interval for a data-0 in Interrogator-to-Tag signaling.

Note 1 to entry: The mnemonic "Tari" derives from the ISO/IEC 18000-61 (Type A) standard, in which Tari is an abbreviation for Type A Reference Interval.

4.75
traceable
not restricting the identifying information a Tag exposes and/or the Tag's operating range

Note 1 to entry: See *untraceable* (4.78).

4.76
untraceable privilege
privilege given to the access password or to a key that grants an Interrogator using the access password or key the right to access untraceably hidden memory and/or to issue an *Untraceable* command

4.77
untraceably hidden memory
memory that an untraceable tag hides from Interrogators with a deasserted untraceable privilege

4.78
untraceable
restricting the identifying information a Tag exposes and/or the Tag's operating range

Note 1 to entry: See *traceable* (4.75).

5 Symbols, abbreviated terms and notation

The principal symbols and abbreviated terms used in this protocol are detailed in ISO/IEC 19762. Symbols, abbreviated terms, and notation specific to this part of ISO/IEC 18000 are as given in 5.1, 5.2 and 5.3, respectively.

5.1 Symbols

BAP	battery assisted passive
BLF	backscatter-link frequency (BLF = $1/T_{pri}$ = DR/TRcal)
C	computed-response indicator
CSI	cryptographic suite identifier
DR	divide ratio
F	file-services indicator (whether a Tag supports the *FileOpen* command)
FrT	frequency tolerance
FT	frequency tolerance
H	hazmat indicator
INACT_T	inactivity threshold
K	killable indicator
M	number of subcarrier cycles per symbol
M_h	RF signal envelope ripple (overshoot)
M_{hh}	FHSS signal envelope ripple (overshoot)
M_{hl}	FHSS signal envelope ripple (undershoot)
M_{hs}	FHSS signal level during a hop
M_l	RF signal envelope ripple (undershoot)
M_s	RF signal level when OFF
NR	nonremovable indicator
Q	slot-count parameter. *Q* is an integer in the range (0,15)
R=>T	interrogator-to-tag (reader-to-tag)
RTcal	interrogator-to-tag (reader-to-tag) calibration symbol
S	security-services indicator (whether a Tag supports the *Challenge* and/or *Authenticate* commands)
SLI	SL indicator
T	numbering system identifier
T=>R	tag-to-Interrogator (tag-to-reader)
T_1	time from Interrogator transmission to Tag response for an *immediate* Tag reply
T_2	time from tag response to Interrogator transmission
T_3	time an Interrogator waits, after T_1, before it issues another command
T_4	minimum time between Interrogator commands

T_5	time from Interrogator transmission to Tag response for a *delayed* Tag reply
T_6	time from Interrogator transmission to first Tag response for an *in-process* Tag reply
T_7	time between Tag responses for an *in-process* Tag reply
Tari	reference time interval for a data-0 in Interrogator-to-tag signalling
T_f or $T_{f,10\text{-}90\%}$	RF signal envelope fall time
Tf	fall time
T_{hf}	FHSS signal envelope fall time
T_{hr}	FHSS signal envelope rise time
T_{hs}	time for an FHSS signal to settle to within a specified percentage of its final value
TN	tag-notification indicator
T_{pri}	backscatter-link pulse-repetition interval (T_{pri} = 1/BLF = TRcal/DR)
T_r or $T_{r,10\text{-}90\%}$	RF signal envelope rise time
Tr	rise time
TRcal	tag-to-Interrogator (tag-to-reader) calibration symbol
T_s	time for an RF signal to settle to within a specified percentage of its final value
U	untraceability indicator
UII	unique item identifier
UMI	user-memory indicator
X	XTID indicator (whether a Tag implements an XTID)
XEB	XPC_W2 indicator
x_{fp}	floating-point value
XI	XPC_W1 indicator
XPC	extended protocol control
$xxxx_2$	binary notation
$xxxx_h$	hexadecimal notation

5.2 Abbreviated terms

AC	activation code
AFI	application family identifier
AM	amplitude modulation
AMSK	activation Mask
ASF	application sub family

ASIC	application specific integrated circuit
ASK	amplitude shift keying
BAM	battery assisted mode
BAT	battery assisted tag
ciphertext	information that is cover-coded
CRC	cyclic redundancy check
CRC-16	sixteen bit CRC
CRC-5	five bit CRC
CW	continuous wave
dBch	decibels referenced to the integrated power in the reference channel
DBR	dead battery response
DSB	double sideband
DSB-ASK	double-sideband amplitude-shift keying
DSFID	data storage format identifier
DSSS	direct sequence spread spectrum
EOF	end of frame
EPC™	electronic product code
ETSI	European Telecommunications Standards Institute
FCC	Federal Communications Commission
FDM	frequency-Division Multiplexing
FHSS	frequency hopping spread spectrum
Handle	16-bit tag identifier
ITF	interrogator-talks-first (reader talks first)

NOTE The common usage is RTF (Reader-talks-first) but the more precise term is ITF, which is used throughout this part of ISO/IEC 18000.

LSB	least significant bit
MAC	message authentication code
MIIM	mobile Item Identication Management
MSB	most significant bit
NoS	number of sensors
NRZ	non return to zero
NSI	numbering system identifier

OTP	one-time programmable
PC	protocol control
PIE	pulse interval encoding
Pivot	decision threshold differentiating an R=>T data-0 symbol from a data-1 symbol
Plaintext	information that is not cover-coded
PM	passive mode
ppm	parts per million
PR-ASK	phase-reversal amplitude shift keying
PSK	phase shift keying or phase shift keyed
R/W	read/write
RF	radio frequency
RFID	radio-frequency identification
RFU	reserved for future use
RN16	16-bit random or pseudo-random number
RNG	random or pseudo-random number generator
RO	read only
RTC	real time clock
SDS	sensor directory system
SOF	start of frame
SS	simple sensor
SSB	single-sideband
SSB-ASK	single-sideband amplitude-shift keying
SSD	simple sensor data
TDM	time-division multiplexing or time-division multiplexed (as appropriate)
TEDS	transducer electronic data sheet
TID	tag-identification or tag identifier, depending on context
UMI	user-memory indicator
UTC	universal coordinated time
WC	wildcard
Word	16 bits
XEB	XPC_W2 indicator

XI	XPC_W1 indicator
XPC	extended protocol control
XPC_W1	XPC word 1
XPC_W2	XPC word 2
XTID	extended tag identifier

5.3 Notation

This part of ISO/IEC 18000 uses the following notational conventions for Type C.

— States and flags are denoted in bold. Some command parameters are also flags; a command parameter used as a flag will be bold. Example: **ready**.

— Command parameters are underlined. Some flags are also command parameters; a flag used as a command parameter will be underlined. Example: Pointer.

— Commands are denoted in italics. Variables are also denoted in italics. Where there might be confusion between commands and variables, this protocol will make an explicit statement. Example: *Query*.

— For logical negation, labels are preceded by '~'. Example: If **flag** is true, then **~flag** is false.

— The symbol, R=>T, refers to commands or signaling from an Interrogator to a Tag (Reader-to-Tag).

— The symbol, T=>R, refers to commands or signaling from a Tag to an Interrogator (Tag-to-Reader).

— The term "address" is used as a synonym for "bit address". All addresses specified in the Clauses denoted to Type C are bit-addresses, unless specified differently (e.g. by adding an according prefix such as "word" in "word address"). The same applies for figures, where all shown addresses are to be interpreted as bit-addresses, unless specified differently.

— The term "word" always refers to a 16-bit word.

6 Protocol requirements - Type C

6.1 Protocol overview

6.1.1 Physical layer

An Interrogator sends information to one or more Tags by modulating an RF carrier using double-sideband amplitude shift keying (DSB-ASK), single-sideband amplitude shift keying (SSB-ASK), or phase-reversal amplitude shift keying (PR-ASK) using a pulse-interval encoding (PIE) format. Tags receive their operating energy from this same modulated RF carrier.

An Interrogator receives information from a Tag by transmitting an unmodulated RF carrier and listening for a backscattered reply. Tags communicate information by backscatter modulating the amplitude and/or phase of the RF carrier. The encoding format, selected in response to Interrogator commands, is either FM0 or Miller-modulated subcarrier. The communications link between Interrogators and Tags is half-duplex, meaning that Tags shall not be required to demodulate Interrogator commands while backscattering. A Tag shall not respond to a mandatory or optional command using full-duplex communications.

6.1.2 Tag-identification layer

An Interrogator manages Tag populations using three basic operations:

a) **Select.** Choosing a Tag population. An Interrogator may use a *Select* command to select one or more Tags based on a value or values in Tag memory, and may use a *Challenge* command to challenge one or more Tags based on Tag support for the desired cryptographic suite and authentication type. An Interrogator may subsequently inventory and access the chosen Tag(s).

b) **Inventory.** Identifying individual Tags. An Interrogator begins an inventory round by transmitting a *Query* command in one of four sessions. One or more Tags may reply. The Interrogator detects a single Tag reply and requests the Tag's UII. Inventory comprises multiple commands. An inventory round operates in one and only one session at a time.

c) **Access.** Communicating with an identified Tag. The Interrogator may perform a core operation such as reading, writing, locking, or killing the Tag; a security-related operation such as authenticating the Tag; or a file-related operation such as opening a particular file in the Tag's User memory. Access comprises multiple commands. An Interrogator may only access a uniquely identified Tag.

6.2 Protocol parameters

6.2.1 Signaling – Physical and media access control parameters

Table 6.1 and Table 6.2 provide an overview of parameters for R=>T and T=>R communications according to this protocol. For those parameters that do not apply to or are not used in this protocol the notation "N/A" indicates that the parameter is "Not Applicable".

Table 6.1 — Interrogator-to-Tag (R=>T) communications

Ref.	Parameter Name	Description
Int:1	Operating Frequency Range	860 – 960 MHz, as required by local regulations
Int:1a	Default Operating Frequency	Determined by local radio regulations and by the radio-frequency environment at the time of the communication
Int:1b	Operating Channels (spread-spectrum systems)	In accordance with local regulations; if the channelization is unregulated, then as specified
Int:1c	Operating Frequency Accuracy	As specified
Int:1d	Frequency Hop Rate (frequency-hopping [FHSS] systems)	In accordance with local regulations
Int:1e	Frequency Hop Sequence (frequency-hopping [FHSS] systems)	In accordance with local regulations
Int:2	Occupied Channel Bandwidth	In accordance with local regulations
Int:2a	Minimum Receiver Bandwidth	In accordance with local regulations
Int:3	Interrogator Transmit Maximum EIRP	In accordance with local regulations
Int:4	Interrogator Transmit Spurious Emissions	As specified; local regulation may impose tighter emission limits
Int:4a	Interrogator Transmit Spurious Emissions, In-Band (spread-spectrum systems)	As specified; local regulation may impose tighter emission limits
Int:4b	Interrogator Transmit Spurious Emissions, Out-of-Band	As specified; local regulation may impose tighter emission limits
Int:5	Interrogator Transmitter Spectrum Mask	As specified; local regulation may impose tighter emission limits
Int:6	Timing	As specified

Table 6.1 *(continued)*

Ref.	Parameter Name	Description
Int:6a	Transmit-to-Receive Turn-Around Time	MAX(RTcal,10T_{pri}) nominal
Int:6b	Receive-to-Transmit Turn-Around Time	3T_{pri} minimum; 20T_{pri} maximum when Tag is in **reply** & **acknowledged** states; no limit otherwise
Int:6c	Dwell Time or Interrogator Transmit Power-On Ramp	1500 µs, maximum settling time
Int:6d	Decay Time or Interrogator Transmit Power-Down Ramp	500 µs, maximum
Int:7	Modulation	DSB-ASK, SSB-ASK, or PR-ASK
Int:7a	Spreading Sequence (direct-sequence [DSSS] systems)	N/A
Int:7b	Chip Rate (spread-spectrum systems)	N/A
Int:7c	Chip Rate Accuracy (spread-spectrum systems)	N/A
Int:7d	Modulation Depth	90% nominal
Int:7e	Duty Cycle	48% – 82.3% (time the waveform is high)
Int:7f	FM Deviation	N/A
Int:8	Data Coding	PIE
Int:9	Bit Rate	26.7 kbps to 128 kbps (assuming equiprobable data)
Int:9a	Bit Rate Accuracy	+/– 1% minimum
Int:10	Interrogator Transmit Modulation Accuracy	As specified
Int:11	Preamble	Required
Int:11a	Preamble Length	As specified
Int:11b	Preamble Waveform(s)	As specified
Int:11c	Bit Sync Sequence	None
Int:11d	Frame Sync Sequence	Required
Int:12	Scrambling (spread-spectrum systems)	N/A
Int:13	Bit Transmission Order	MSB is transmitted first
Int:14	Wake-up Process	As specified
Int:15	Polarization	Not specified

Table 6.2 — Tag-to-Interrogator (T=>R) communications

Ref.	Parameter Name	Description
Tag:1	Operating Frequency Range	860 – 960 MHz, inclusive
Tag:1a	Default Operating Frequency	Tags respond to Interrogator signals that satisfy Int:1a
Tag:1b	Operating Channels (spread-spectrum systems)	Tags respond to Interrogator signals that satisfy Int:1b
Tag:1c	Operating Frequency Accuracy	As specified
Tag:1d	Frequency Hop Rate (frequency-hopping [FHSS] systems)	Tags respond to Interrogator signals that satisfy Int:1d

Table 6.2 *(continued)*

Ref.	Parameter Name	Description
Tag:1e	Frequency Hop Sequence (frequency-hopping [FHSS] systems)	Tags respond to Interrogator signals that satisfy Int:1e
Tag:2	Occupied Channel Bandwidth	In accordance with local regulations
Tag:3	Transmit Maximum EIRP	In accordance with local regulations
Tag:4	Transmit Spurious Emissions	In accordance with local regulations
Tag:4a	Transmit Spurious Emissions, In-Band (spread-spectrum systems)	In accordance with local regulations
Tag:4b	Transmit Spurious Emissions, Out-of-Band	In accordance with local regulations
Tag:5	Transmit Spectrum Mask	In accordance with local regulations
Tag:6a	Transmit-to-Receive Turn-Around Time	$3T_{pri}$ minimum, $32T_{pri}$ maximum in **reply** & **acknowledged** states; no limit otherwise
Tag:6b	Receive-to-Transmit Turn-Around Time	MAX(RTcal,$10T_{pri}$) nominal
Tag:6c	Dwell Time or Transmit Power-On Ramp	Receive commands 1500 µs after power-up
Tag:6d	Decay Time or Transmit Power-Down Ramp	N/A
Tag:7	Modulation	ASK and/or PSK modulation (selected by Tag)
Tag:7a	Spreading Sequence (direct sequence [DSSS] systems)	N/A
Tag:7b	Chip Rate (spread-spectrum systems)	N/A
Tag:7c	Chip Rate Accuracy (spread-spectrum systems)	N/A
Tag:7d	On-Off Ratio	Not specified
Tag:7e	Subcarrier Frequency	40 kHz to 640 kHz
Tag:7f	Subcarrier Frequency Accuracy	As specified
Tag:7g	Subcarrier Modulation	Miller, at the data rate
Tag:7h	Duty Cycle	FM0: 50%, nominal Subcarrier: 50%, nominal
Tag:7I	FM Deviation	N/A
Tag:8	Data Coding	Baseband FM0 or Miller-modulated subcarrier (selected by the Interrogator)
Tag:9	Bit Rate	FM0: 40 kbps to 640 kbps Subcarrier modulated: 5 kbps to 320 kbps
Tag:9a	Bit Rate Accuracy	Same as Subcarrier Frequency Accuracy; see Tag:7f
Tag:10	Tag Transmit Modulation Accuracy (frequency-hopping [FHSS] systems	N/A
Tag:11	Preamble	Required
Tag:11a	Preamble Length	As specified
Tag:11b	Preamble Waveform	As specified
Tag:11c	Bit-Sync Sequence	None
Tag:11d	Frame-Sync Sequence	None

Table 6.2 *(continued)*

Ref.	Parameter Name	Description
Tag:12	Scrambling (spread-spectrum systems)	N/A
Tag:13	Bit Transmission Order	MSB is transmitted first
Tag:14	Reserved	Deliberately left blank
Tag:15	Polarization	Tag dependent; not specified by this protocol
Tag:16	Minimum Tag Receiver Bandwidth	Tag dependent; not specified by this protocol.

6.2.2 Logical – Operating procedure parameters

Table 6.3 and Table 6.4 identify and describe parameters used by an Interrogator during the selection, inventory, and access of Tags according to this protocol. For those parameters that do not apply to or are not used in this protocol the notation "N/A" indicates that the parameter is "Not Applicable".

Table 6.3 — Tag inventory and access parameters

Ref.	Parameter Name	Description
P:1	Who Talks First	Interrogator
P:2	Tag Addressing Capability	As specified
P:3	Tag UII	Contained in Tag memory
P:3a	UII Length	As specified
P:3b	UII Format	**T**=0_2: As specified in the GS1 EPC Tag Data Standard, **T**=1_2: As specified in ISO/IEC 15961
P:4	Read size	Multiples of 16 bits
P:5	Write Size	Multiples of 16 bits
P:6	Read Transaction Time	Varied with R=>T & T=>R link rate and number of bits being read
P:7	Write Transaction Time	20 ms (maximum) after end of *Write* command
P:8	Error Detection	Interrogator-to-Tag: *Select* and *Challenge* commands: 16-bit CRC *Query* command: 5-bit CRC Other Inventory commands: Command length Access commands: 16-bit CRC Tag-to-Interrogator: PC/XPC, UII: 16-bit CRC RN16: None or 16-bit CRC (varies by command) handle: 16-bit CRC All other: 16-bit CRC
P:9	Error Correction	None
P:10	Memory Size	Tag dependent, extensible (size is neither limited nor specified by this protocol)
P:11	Command Structure and Extensibility	As specified

Table 6.4 — Collision management parameters

Ref.	Parameter Name	Description
A:1	Type (Probabilistic or Deterministic)	Probabilistic
A:2	Linearity	Linear up to 2^{15} Tags in the Interrogator's RF field; above that number, NlogN for Tags with unique UIIs
A:3	Tag Inventory Capacity	Unlimited for Tags with unique UIIs

6.3 Description of operating procedure

The operating procedure defines the physical and logical requirements for an Interrogator-talks-first, random-slotted collision arbitration, RFID system operating in the 860 – 960 MHz frequency range.

6.3.1 Physical interface

The physical interface between an Interrogator and a Tag may be viewed as the signaling layer in a layered network communication system. The signaling interface defines frequencies, modulation, data coding, RF envelope, data rates, and other parameters required for RF communications.

6.3.1.1 Operational frequencies

Tags shall receive power from and communicate with Interrogators within the frequency range from 860 – 960 MHz, inclusive. An Interrogator's choice of operational frequency will be determined by local radio regulations and by the local radio-frequency environment. Interrogators certified for operation in dense-Interrogator environments shall support, but are not required to always use, the optional dense-Interrogator mode described in Annex G.

6.3.1.2 Interrogator-to-Tag (R=>T) communications

An Interrogator communicates with one or more Tags by modulating an RF carrier using DSB-ASK, SSB-ASK, or PR-ASK with PIE encoding. Interrogators shall use a fixed modulation format and data rate for the duration of an inventory round, where "inventory round" is defined in 4. The Interrogator sets the data rate by means of the preamble that initiates the inventory round.

The high values in Figure 6.1 Figure 6.2, Figure 6.3, Figure 6.4, and Figure 6.5 correspond to emitted CW (i.e. an Interrogator delivering power to the Tag or Tags) whereas the low values correspond to attenuated CW.

6.3.1.2.1 Interrogator frequency accuracy

Interrogators certified for operation in single- or multiple-Interrogator environments shall have a frequency accuracy that meets local regulations.

Interrogators certified for operation in dense-Interrogator environments shall have a frequency accuracy of +/– 10 ppm over the nominal temperature range (–25°C to +40°C) and +/– 20 ppm over the extended temperature range (–40°C to +65°C). Interrogators rated by the manufacturer to have a temperature range wider than nominal but different from extended shall have a frequency accuracy of +/– 10 ppm over the nominal temperature range and +/– 20 ppm to the extent of their rated range. If local regulations specify tighter frequency accuracy then the Interrogator shall meet the local regulations.

6.3.1.2.2 Modulation

Interrogators shall communicate using DSB-ASK, SSB-ASK, or PR-ASK modulation, detailed in Annex H. Tags shall demodulate all three modulation types.

6.3.1.2.3 Data encoding

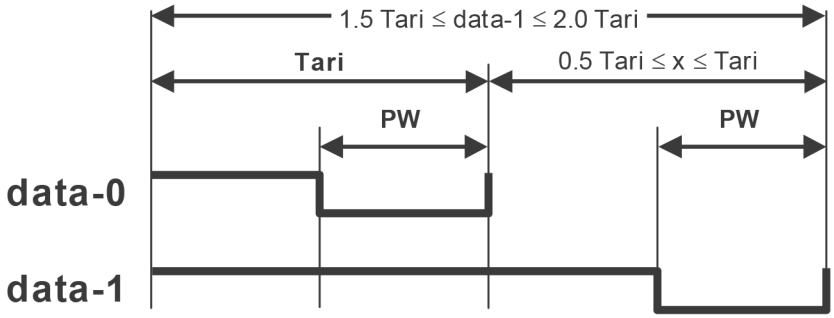


Figure 6.1 — PIE symbols

The R=>T link shall use PIE, shown in Figure 6.1. Tari is the reference time interval for Interrogator-to-Tag signaling, and is the duration of a data-0. High values represent transmitted CW; low values represent attenuated CW. Pulse modulation depth, rise time, fall time, and PW shall be as specified in Table 6.5, and shall be the same for a data-0 and a data-1. Interrogators shall use a fixed modulation depth, rise time, fall time, PW, Tari, data-0 length, and data-1 length for the duration of an inventory round. The RF envelope shall be as specified in Figure 6.2.

6.3.1.2.4 Tari values

Interrogators shall communicate using Tari values in the range of 6.25µs to 25µs. Interrogator compliance shall be evaluated using at least one Tari value between 6.25µs and 25µs with at least one value of the parameter *x*. The tolerance on all parameters specified in units of Tari shall be +/–1%. The choice of Tari value and *x* shall be in accordance with local radio regulations.

6.3.1.2.5 R=>T RF envelope

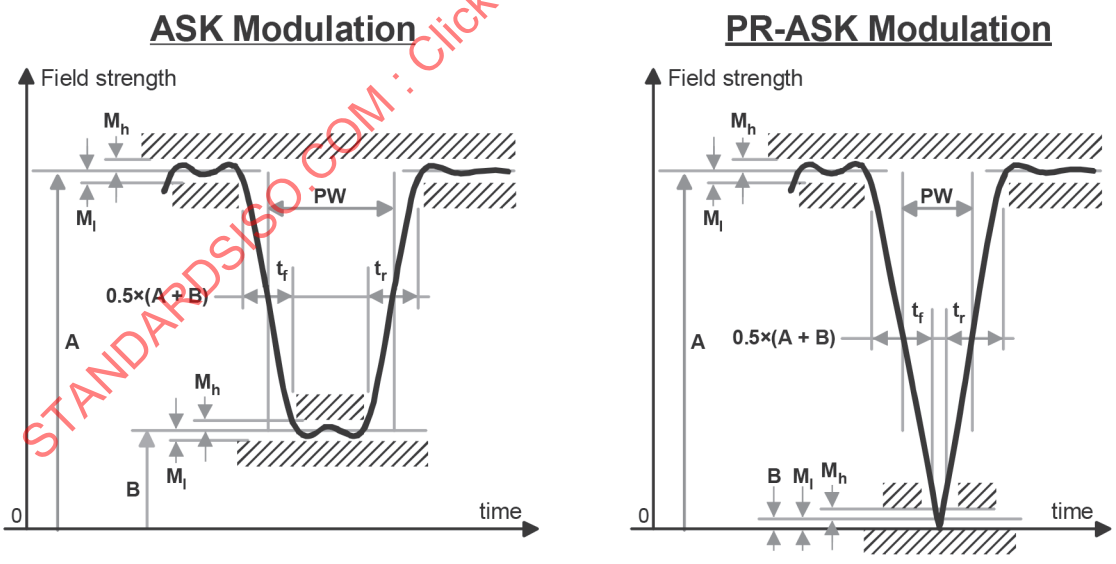


Figure 6.2 — Interrogator-to-Tag RF envelope

Table 6.5 — RF envelope parameters

Tari	Parameter	Symbol	Minimum	Nominal	Maximum	Units
6.25 µs to 25 µs	Modulation Depth	(A–B)/A	80	90	100	%
	RF Envelope Ripple	$M_h = M_l$	0		0.05(A–B)	V/m or A/m
	RF Envelope Rise Time	$t_{r,10-90\%}$	0		0.33Tari	µs
	RF Envelope Fall Time	$t_{f,10-90\%}$	0		0.33Tari	µs
	RF Pulsewidth	PW	MAX(0.265Tari, 2)		0.525Tari	µs

The R=>T RF envelope shall comply with Figure 6.2 and Table 6.5. The electric or magnetic field strength A (as appropriate) is the maximum amplitude of the RF envelope, measured in units of V/m or A/m, respectively. Tari is defined in Figure 6.1. The pulsewidth is measured at the 50% point on the pulse. An Interrogator shall not change the R=>T modulation type (i.e. shall not switch between DSB-ASK, SSB-ASK, or PR-ASK) without first powering down its RF waveform (see 6.3.1.2.7).

6.3.1.2.6 Interrogator power-up waveform

The Interrogator power-up RF envelope shall comply with Figure 6.3 and Table 6.6. Once the carrier level has risen above the 10% level, the power-up envelope shall rise monotonically until at least the ripple limit M_l. The RF envelope shall not fall below the 90% point in Figure 6.3 during interval T_s. Interrogators shall not issue commands before the end of the maximum settling-time interval in Table 6.6 (i.e. before the end of T_s). Interrogators shall meet the frequency-accuracy requirement specified in 6.3.1.2.1 by the end of interval T_s in Figure 6.3.

6.3.1.2.7 Interrogator power-down waveform

The Interrogator power-down RF envelope shall comply with Figure 6.3 and Table 6.7. Once the carrier level has fallen below the 90% level, the power-down envelope shall fall monotonically until the power-off limit M_s. Once powered off, an Interrogator shall remain powered off for a least 1ms before powering up again.

6.3.1.2.8 R=>T preamble and frame-sync

An Interrogator shall begin all R=>T signaling with either a preamble or a frame-sync, both of which are shown in Figure 6.4. A preamble shall precede a *Query* command (see 6.3.2.12.2.1) and denotes the start of an inventory round. All other signaling shall begin with a frame-sync. The tolerance on all parameters specified in units of Tari shall be +/–1%. PW shall be as specified in Table 6.5. The RF envelope shall be as specified in Figure 6.2.

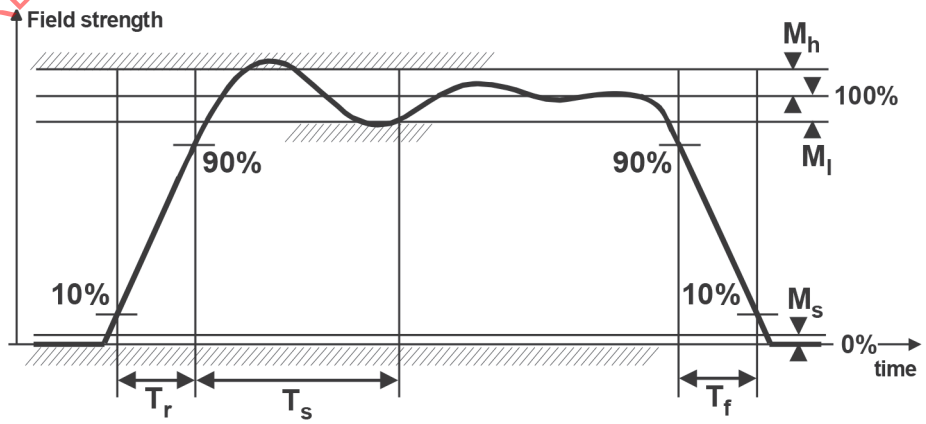


Figure 6.3 — Interrogator power-up and power-down RF envelope

Table 6.6 — Interrogator power-up waveform parameters

Parameter	Definition	Minimum	Nominal	Maximum	Units
T_r	Rise time	1		500	µs
T_s	Settling time			1500	µs
M_s	Signal level when OFF			1	% full scale
M_l	Undershoot			5	% full scale
M_h	Overshoot			5	% full scale

Table 6.7 — Interrogator power-down waveform parameters

Parameter	Definition	Minimum	Nominal	Maximum	Units
T_f	Fall time	1		500	µs
M_s	Signal level when OFF			1	% full scale
M_l	Undershoot			5	% full scale
M_h	Overshoot			5	% full scale

A preamble shall comprise a fixed-length start delimiter, a data-0 symbol, an R=>T calibration (RTcal) symbol, and a T=>R calibration (TRcal) symbol.

— **RTcal:** An Interrogator shall set RTcal equal to the length of a data-0 symbol plus the length of a data-1 symbol (RTcal = 0_{length} + 1_{length}). A Tag shall measure the length of RTcal and compute *pivot* = RTcal / 2. A Tag shall interpret subsequent Interrogator symbols shorter than *pivot* to be data-0s, and subsequent Interrogator symbols longer than *pivot* to be data-1s. A Tag shall interpret symbols longer than 4 RTcal to be invalid. Prior to changing RTcal, an Interrogator shall transmit CW for a minimum of 8 RTcal.

— **TRcal:** An Interrogator shall specify a Tag's backscatter link frequency (its FM0 datarate or the frequency of its Miller subcarrier) using the TRcal and divide ratio (DR) in the preamble and payload, respectively, of a *Query* command that initiates an inventory round. Equation (1) specifies the relationship between the backscatter link frequency (BLF), TRcal, and DR. A Tag shall measure the length of TRcal, compute BLF, and adjust its T=>R link rate to be equal to BLF (Table 6.9 shows BLF values and tolerances). The TRcal and RTcal that an Interrogator uses in any inventory round shall meet the constraints in Equation (2):

$$BLF = \frac{DR}{TRcal} \quad (1)$$

$$1.1 \times RTcal \leq TRcal \leq 3 \times RTcal \quad (2)$$

A frame-sync is identical to a preamble, minus the TRcal symbol. An Interrogator, for the duration of an inventory round, shall use the same length RTcal in a frame-sync as it used in the preamble that initiated the round.

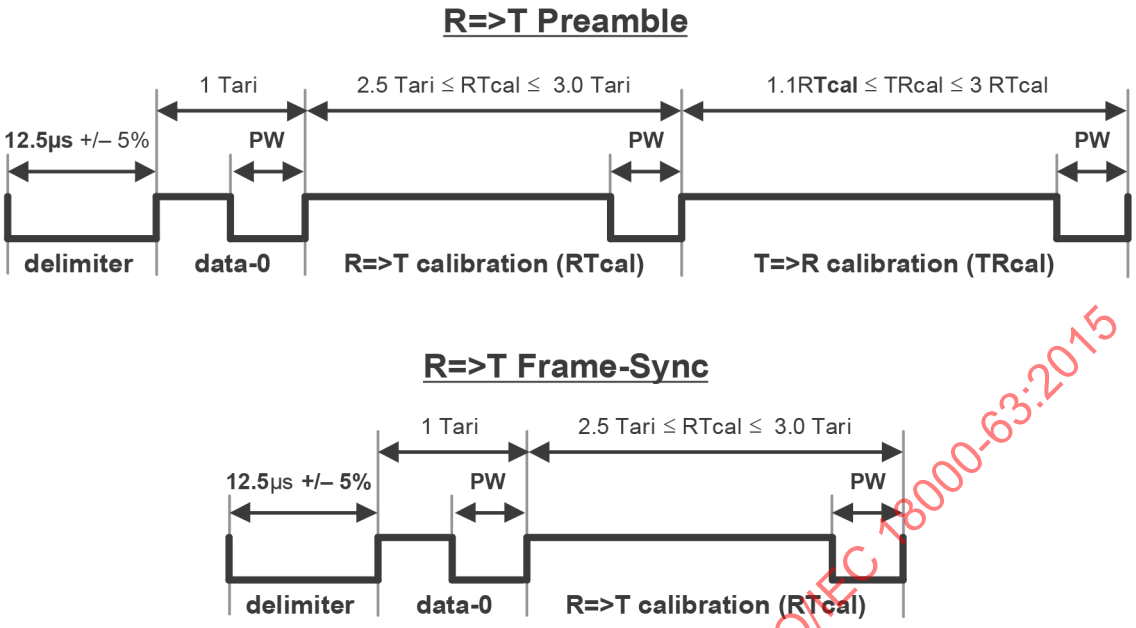


Figure 6.4 — R=>T preamble and frame-sync

6.3.1.2.9 Frequency-hopping spread-spectrum waveform

When an Interrogator uses frequency-hopping spread spectrum (FHSS) signaling, the Interrogator's RF envelope shall comply with Figure 6.5 and Table 6.8. The RF envelope shall not fall below the 90% point in Figure 6.5 during interval T_{hs}. Interrogators shall not issue commands before the end of the maximum settling-time interval in Table 6.8 (i.e. before the end of T_{hs}). The maximum time between frequency hops and the minimum RF-off time during a hop shall meet local regulatory requirements. Interrogators shall meet the frequency-accuracy requirement specified in 6.3.1.2.1 by the end of interval T_{hs} in Figure 6.5.

6.3.1.2.10 Frequency-hopping spread-spectrum channelization

Interrogators certified for operation in single-Interrogator environments shall meet local regulations for spread-spectrum channelization. Interrogators certified for operation in multiple- or dense-Interrogator environments shall meet local regulations for spread-spectrum channelization, unless the channelization is unregulated. In which case Interrogators should adopt a channel plan that fits for the chosen regulatory region (see also Annex G, which describes multiple- and dense-Interrogator channelized signaling).

6.3.1.2.11 Transmit mask

Interrogators certified for operation according to this protocol shall meet local regulations for out-of-channel and out-of-band spurious radio-frequency emissions.

Interrogators certified for operation in multiple-Interrogator environments shall meet both local regulations and the Multiple-Interrogator Transmit Mask described below and shown in Figure 6.6.

Multiple-Interrogator Transmit Mask: For an Interrogator transmitting random data in channel R, and any other channel $S \neq R$, the ratio of the integrated power $P()$ in channel S to that in channel R shall not exceed the specified values:

— $|R - S| = 1$: $10\log_{10}(P(S) / P(R)) < -20$ dB

— $|R - S| = 2$: $10\log_{10}(P(S) / P(R)) < -50$ dB

— $|R - S| = 3$: $10\log_{10}(P(S) / P(R)) < -60$ dB

— $|R - S| > 3$: $10\log_{10}(P(S) / P(R)) < -65$ dB

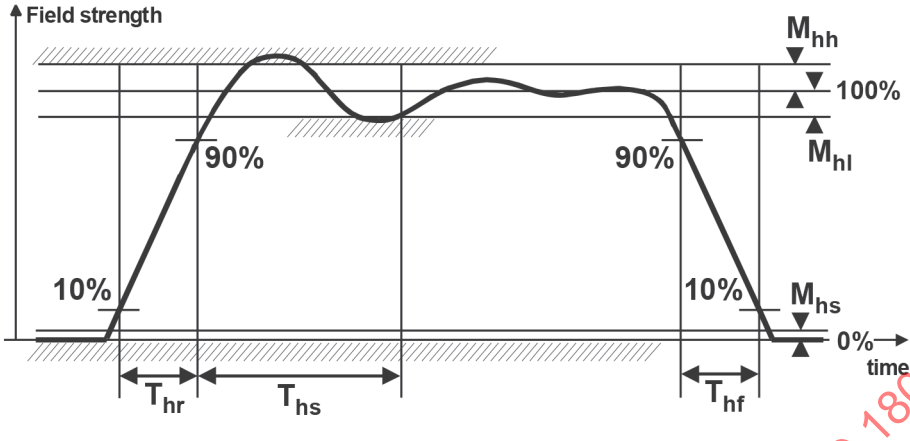


Figure 6.5 — FHSS Interrogator RF envelope

Table 6.8 — FHSS waveform parameters

Parameter	Definition	Minimum	Nominal	Maximum	Units
T_{hr}	Rise time			500	µs
T_{hs}	Settling time			1500	µs
T_{hf}	Fall time			500	µs
M_{hs}	Signal level during hop			1	% full scale
M_{hl}	Undershoot			5	% full scale
M_{hh}	Overshoot			5	% full scale

Where $P()$ denotes the total integrated power in the specified channel. This mask is shown graphically in Figure 6.6, with dBch defined as dB referenced to the integrated power in the reference channel. The channel width shall be as specified by local regulations, unless the width is unregulated, in which case Interrogators should adopt a suitable width for the chosen regulatory region. The channel spacing shall be set equal to the channel width (measured channel center to channel center). For any transmit channel R, two exceptions to the mask are permitted, provided that

— neither exception exceeds –50 dBch, and

— neither exception exceeds local regulatory requirements.

An exception occurs when the integrated power in a channel S exceeds the mask. Each channel that exceeds the mask shall be counted as an exception.

Interrogators certified for operation in dense-Interrogator environments shall meet both local regulations and the Dense-Interrogator Transmit Mask described below and shown in Figure 6.7. Interrogators may meet the Dense-Interrogator Transmit Mask during non-dense-Interrogator operation. Regardless of the mask used, Interrogators certified for operation in dense-Interrogator environments shall not be permitted the two exceptions to the transmit mask that are allowed for Interrogators certified for operation in multiple-Interrogator environments.

Dense-Interrogator Transmit Mask: For Interrogator transmissions centered at a frequency f_c, a 2.5/Tari bandwidth R_{BW} also centered at f_c, an offset frequency f_o = 2.5/Tari, and a 2.5/Tari bandwidth S_{BW} centered at $(n \times f_o) + f_c$ (integer n), the ratio of the integrated power $P()$ in S_{BW} to that in R_{BW} with the Interrogator transmitting random data shall not exceed the specified values:

— $|n| = 1$: $10\log_{10}(P(S_{BW}) / P(R_{BW})) < -30$ dB

— $|n| = 2$: $10\log_{10}(P(S_{BW}) / P(R_{BW})) < -60$ dB

— $|n| > 2$: $10\log_{10}(P(S_{BW}) / P(R_{BW})) < -65$ dB

Where $P()$ denotes the total integrated power in the 2.5/Tari reference bandwidth. This mask is shown graphically in Figure 6.7, with dBch defined as dB referenced to the integrated power in the reference channel.

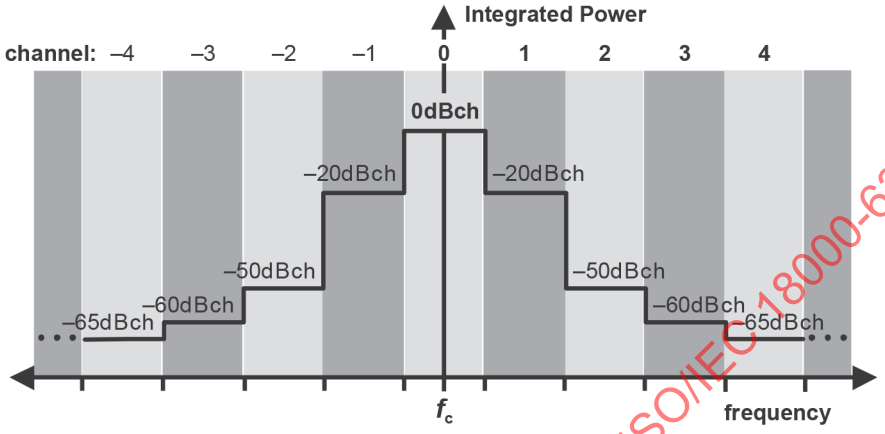


Figure 6.6 — Transmit mask for multiple-Interrogator environments

6.3.1.3 Tag-to-Interrogator (T=>R) communications

A Tag communicates with an Interrogator using backscatter modulation, in which the Tag switches the reflection coefficient of its antenna between two states in accordance with the data being sent.

A Tag shall backscatter using a fixed modulation format, data encoding, and data rate for the duration of an inventory round, where "inventory round" is defined in 4. The Tag selects the modulation format; the Interrogator selects the data encoding and data rate by means of the *Query* command that initiates the round. The low values in Figure 6.9, Figure 6.10, Figure 6.11, Figure 6.13, Figure 6.14, and Figure 6.15 correspond to the antenna-reflectivity state the Tag exhibits during the CW period prior to a T=>R preamble (e.g. ASK Tag absorbing power), whereas the high values correspond to the antenna-reflectivity state the Tag exhibits during the first high pulse of a T=>R preamble (e.g. ASK Tag reflecting power).

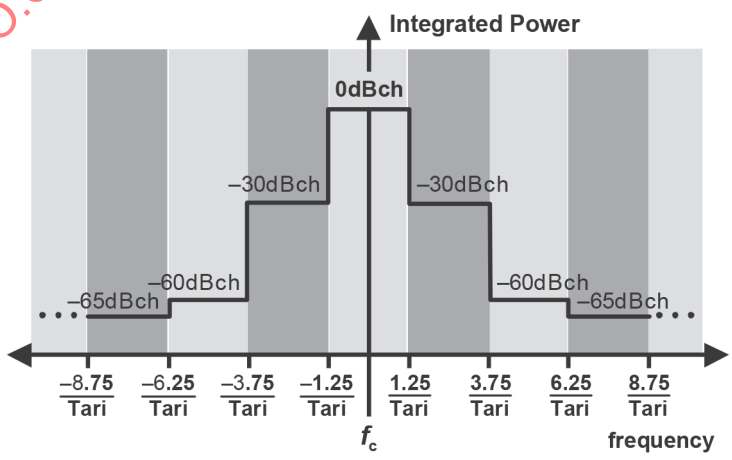


Figure 6.7 — Transmit mask for dense-Interrogator environments

6.3.1.3.1 Modulation

Tag backscatter shall use ASK and/or PSK modulation. The Tag manufacturer selects the modulation format. Interrogators shall demodulate both modulation types.

6.3.1.3.2 Data encoding

Tags shall encode the backscattered data as either FM0 baseband or Miller modulation of a subcarrier at the data rate. The Interrogator specifies the encoding type.

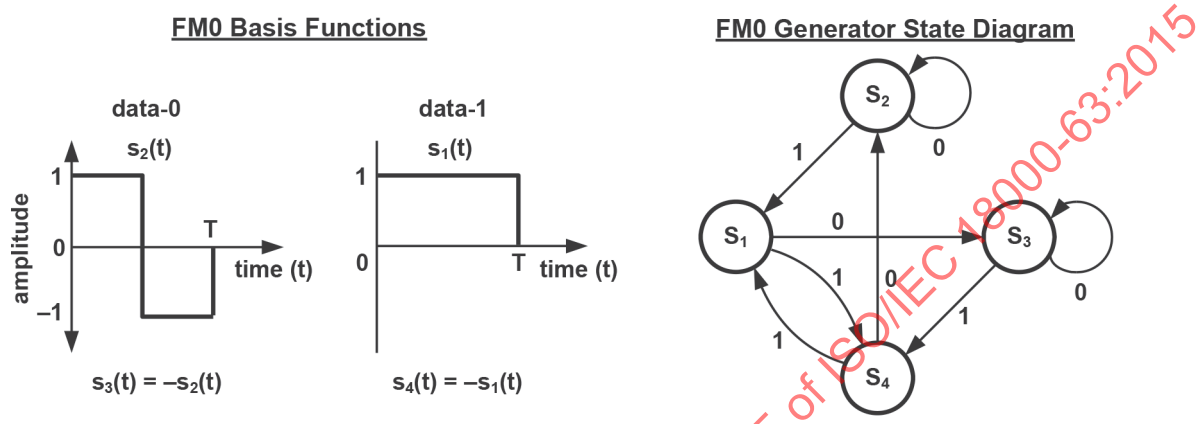


Figure 6.8 — FM0 basis functions and generator state diagram

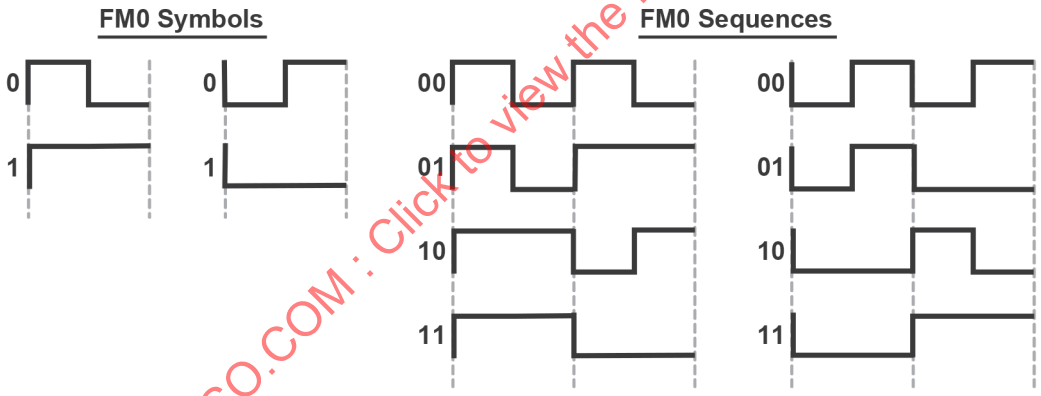


Figure 6.9 — FM0 symbols and sequences

6.3.1.3.2.1 FM0 baseband

Figure 6.8 shows basis functions and a state diagram for generating FM0 (bi-phase space) encoding. FM0 inverts the baseband phase at every symbol boundary; a data-0 has an additional mid-symbol phase inversion. The state diagram in Figure 6.8 maps a logical data sequence to the FM0 basis functions that are transmitted. The state labels, S_1–S_4, indicate four possible FM0-encoded symbols, represented by the two phases of each of the FM0 basis functions. The state labels also represent the FM0 waveform that is transmitted upon entering the state. The labels on the state transitions indicate the logical values of the data sequence to be encoded. For example, a transition from state S_2 to S_3 is disallowed because the resulting transmission would not have a phase inversion on a symbol boundary.

Figure 6.9 shows generated baseband FM0 symbols and sequences. The duty cycle of a 00 or 11 sequence, measured at the modulator output, shall be a minimum of 45% and a maximum of 55%, with a nominal value of 50%. FM0 encoding has memory; consequently, the choice of FM0 sequences in

Figure 6.9 depends on prior transmissions. FM0 signaling shall always end with a "dummy" data-1 bit at the end of a transmission, as shown in Figure 6.10.

6.3.1.3.2.2 FM0 preamble

T=>R FM0 signaling shall begin with one of the two preambles shown in Figure 6.11. The choice depends on the TRext value specified in the *Query* that initiated the inventory round, unless a Tag is replying to a command that uses a *delayed* or *in-process* reply (see 6.3.1.6), in which case a Tag shall use the extended preamble regardless of TRext (i.e. a Tag replies as if TRext=1 regardless of the TRext value specified in the *Query*—see 6.3.2.12.3). The "v" shown in Figure 6.11 indicates an FM0 violation (i.e. a phase inversion should have occurred but did not).

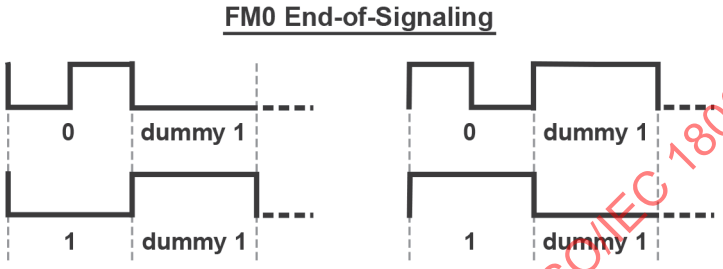


Figure 6.10 — Terminating FM0 transmissions

6.3.1.3.2.3 Miller-modulated subcarrier

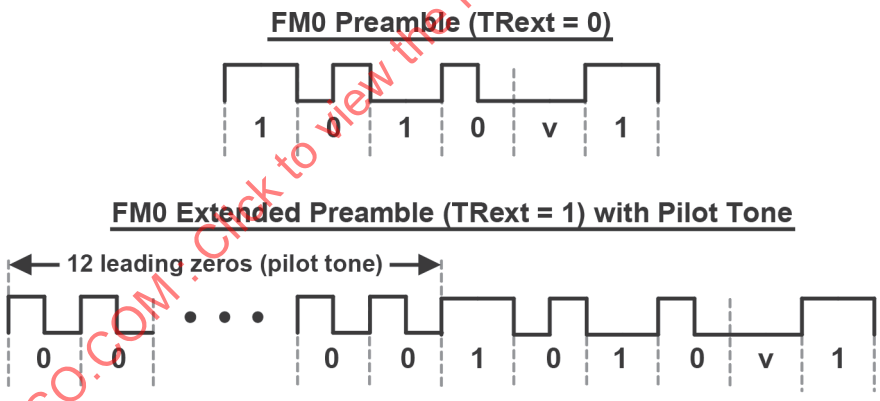


Figure 6.11 — FM0 T=>R preamble

Figure 6.12 shows basis functions and a state diagram for generating Miller encoding. Baseband Miller inverts its phase between two data-0s in sequence. Baseband Miller also places a phase inversion in the middle of a data-1 symbol. The state diagram in Figure 6.12 maps a logical data sequence to baseband Miller basis functions. The state labels, S_1–S_4, indicate four possible Miller-encoded symbols, represented by the two phases of each of the Miller basis functions. The state labels also represent the baseband Miller waveform that is generated upon entering the state. The transmitted waveform is the baseband waveform multiplied by a square-wave at M times the symbol rate. The labels on the state transitions indicate the logical values of the data sequence to be encoded. For example, a transition from state S_1 to S_3 is disallowed because the resulting transmission would have a phase inversion on a symbol boundary between a data-0 and a data-1.

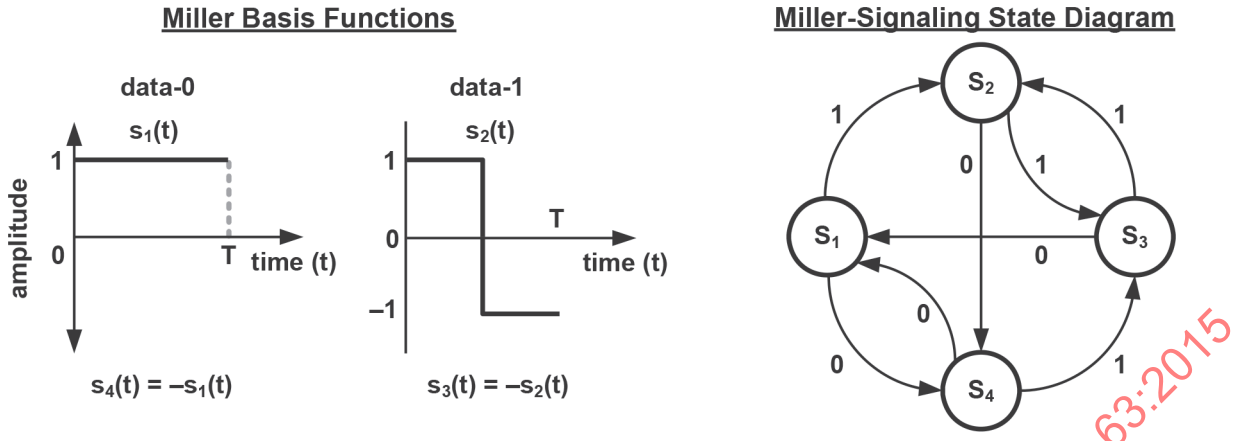


Figure 6.12 — Miller basis functions and generator state diagram

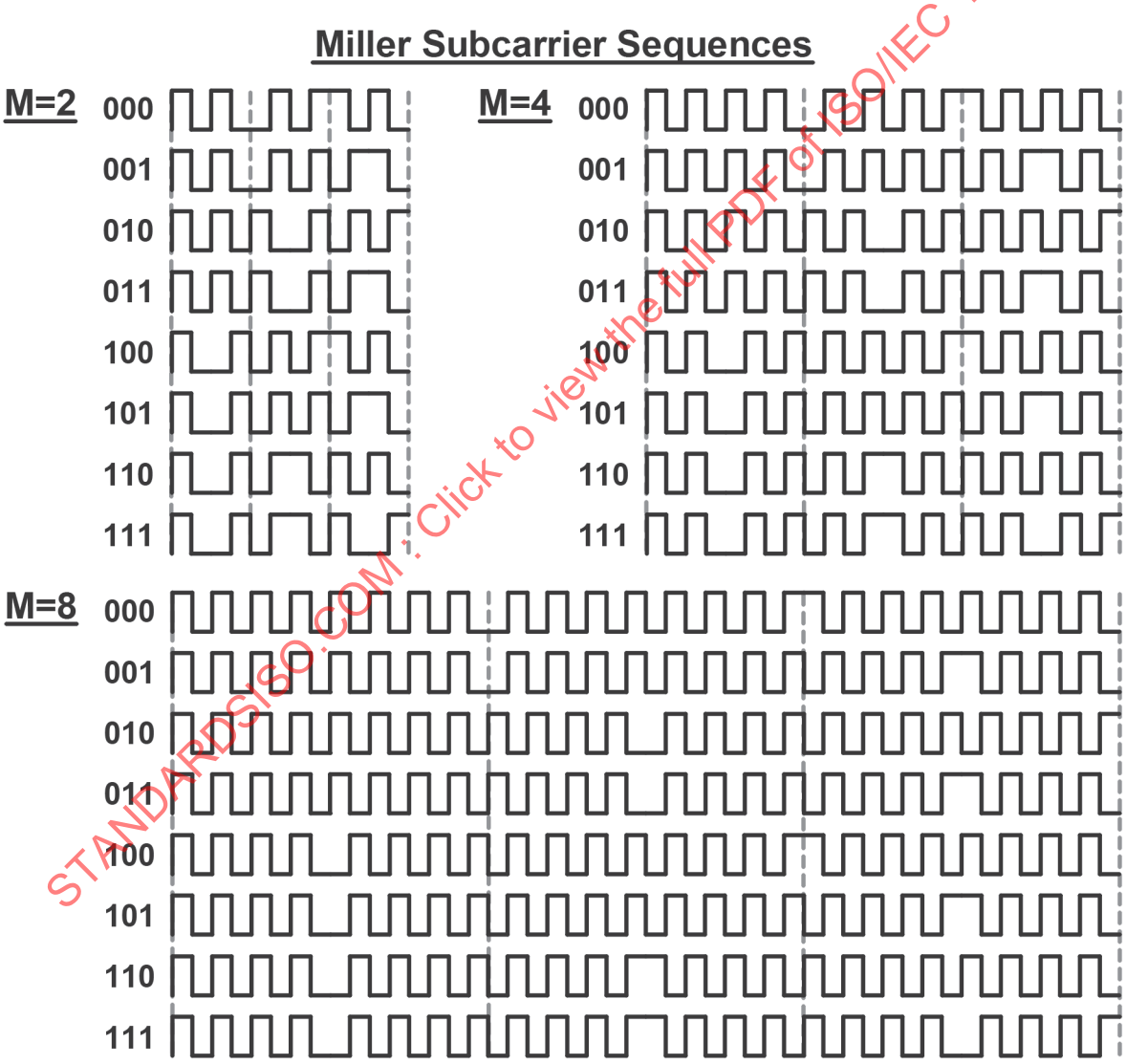


Figure 6.13 — Subcarrier sequences

Figure 6.13 shows Miller-modulated subcarrier sequences; the Miller sequence shall contain exactly two, four, or eight subcarrier cycles per bit, depending on the M value specified in the *Query* command

that initiated the inventory round (see Table 6.10). The duty cycle of a 0 or 1 symbol, measured at the modulator output, shall be a minimum of 45% and a maximum of 55%, with a nominal value of 50%. Miller encoding has memory; consequently, the choice of Miller sequences in Figure 6.13 depends on prior transmissions. Miller signaling shall always end with a "dummy" data-1 bit at the end of a transmission, as shown in Figure 6.14.

6.3.1.3.2.4 Miller subcarrier preamble

T=>R subcarrier signaling shall begin with one of the two preambles shown in Figure 6.15. The choice depends on the TRext value specified in the *Query* that initiated the inventory round, unless a Tag is replying to a command that uses a *delayed* or *in-process* reply (see 6.3.1.6), in which case a Tag shall use the extended preamble regardless of TRext (i.e. a Tag replies as if TRext=1 regardless of the TRext value specified in the *Query*—see 6.3.2.12.3).

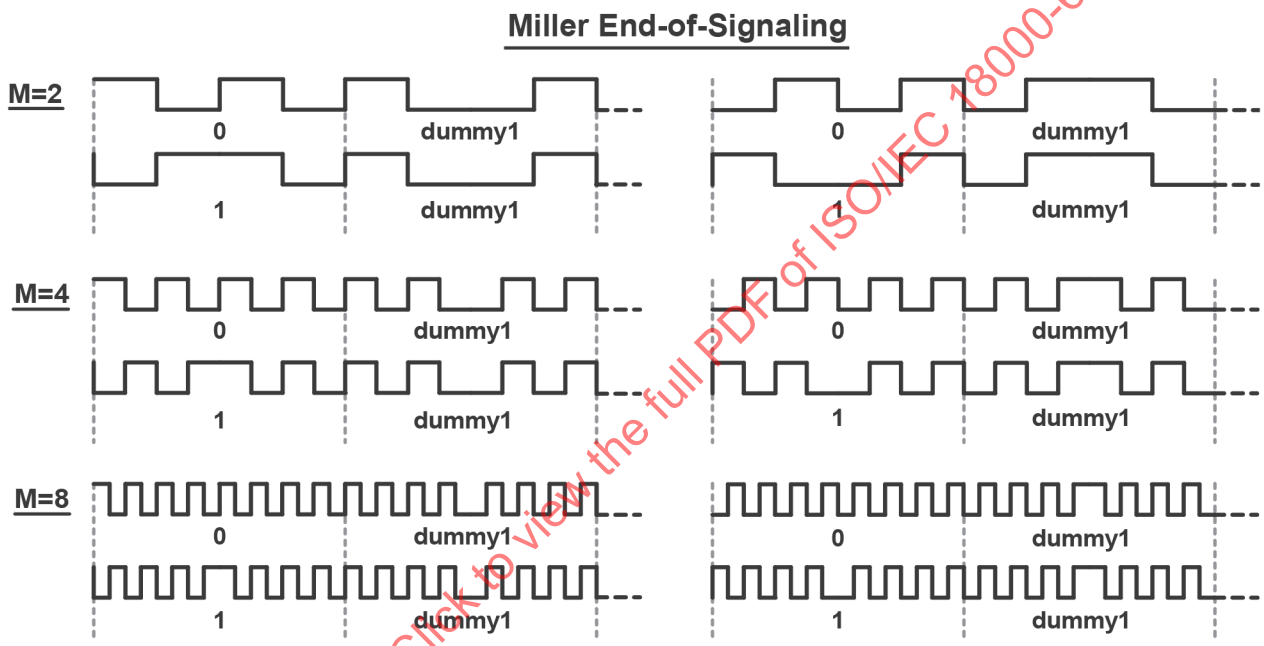


Figure 6.14 — Terminating subcarrier transmissions

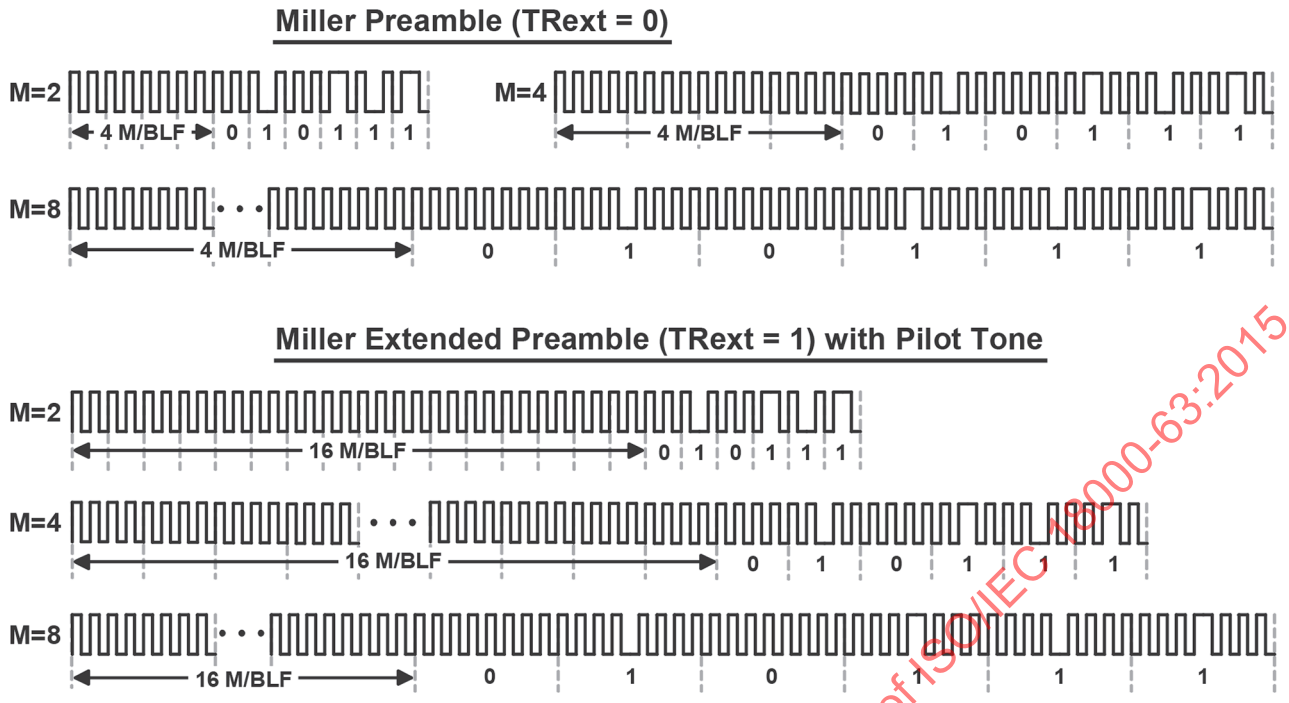


Figure 6.15 — Subcarrier T=>R preamble

6.3.1.3.3 Tag supported Tari values and backscatter link rates

Tags shall support all R=>T Tari values in the range of 6.25µs to 25µs, over all parameters allowed by 6.3.1.2.3.

Tags shall support the T=>R link frequencies and tolerances specified in Table 6.9 and the T=>R data rates specified in Table 6.10. The frequency-variation requirement in Table 6.9 includes both frequency drift and short-term frequency variation during Tag response to an Interrogator command. The *Query* command that initiates an inventory round specifies DR in Table 6.9 and M in Table 6.10; the preamble that precedes the *Query* specifies TRcal. BLF is computed using Eq. (1). These four parameters together define the backscatter frequency, modulation type (FM0 or Miller), and T=>R data rate for the round (see also 6.3.1.2.8).

Table 6.9 — Tag-to-Interrogator link frequencies

DR: Divide Ratio	TRcal [1] (µs +/– 1%)	BLF: Link Frequency (kHz)	Frequency Tolerance FrT (nominal temp)	Frequency Tolerance FrT (extended temp)	Frequency variation during backscatter
64/3	33.3	640	+ / – 15%	+ / – 15%	+ / – 2.5%
	33.3 < TRcal < 66.7	320 < BLF < 640	+ / – 22%	+ / – 22%	+ / – 2.5%
	66.7	320	+ / – 10%	+ / – 15%	+ / – 2.5%
	66.7 < TRcal < 83.3	256 < BLF < 320	+ / – 12%	+ / – 15%	+ / – 2.5%
	83.3	256	+ / – 10%	+ / – 10%	+ / – 2.5%
	83.3 < TRcal ≤ 133.3	160 ≤ BLF < 256	+ / – 10%	+ / – 12%	+ / – 2.5%
	133.3 < TRcal ≤ 200	107 ≤ BLF < 160	+ / – 7%	+ / – 7%	+ / – 2.5%
	200 < TRcal ≤ 225	95 ≤ BLF < 107	+ / – 5%	+ / – 5%	+ / – 2.5%

Table 6.9 *(continued)*

DR: Divide Ratio	Trcal [1] (µs +/– 1%)	BLF: Link Frequency (kHz)	Frequency Tolerance FrT (nominal temp)	Frequency Tolerance FrT (extended temp)	Frequency variation during backscatter
8	17.2 ≤ TRcal < 25	320 < BLF ≤ 465	+ / – 19%	+ / – 19%	+ / – 2.5%
	25	320	+ / – 10%	+ / – 15%	+ / – 2.5%
	25 < TRcal < 31.25	256 < BLF < 320	+ / – 12%	+ / – 15%	+ / – 2.5%
	31.25	256	+ / – 10%	+ / – 10%	+ / – 2.5%
	31.25 < TRcal < 50	160 < BLF < 256	+ / – 10%	+ / – 10%	+ / – 2.5%
	50	160	+ / – 7%	+ / – 7%	+ / – 2.5%
	50 < TRcal ≤ 75	107 ≤ BLF < 160	+ / – 7%	+ / – 7%	+ / – 2.5%
	75 < TRcal ≤ 200	40 ≤ BLF < 107	+ / – 4%	+ / – 4%	+ / – 2.5%

Note 1 Allowing two different TRcal values (with two different DR values) to specify the same BLF offers flexibility in specifying Tari and RTcal.

Table 6.10 — Tag-to-Interrogator data rates

M: Number of subcarrier cycles per symbol	Modulation type	Data rate (kbps)
1	FM0 baseband	BLF
2	Miller subcarrier	BLF/2
4	Miller subcarrier	BLF/4
8	Miller subcarrier	BLF/8

6.3.1.3.4 Tag power-up timing

Tags energized by an Interrogator shall be capable of receiving and acting on Interrogator commands within a period not exceeding the maximum settling-time interval specified in Table 6.6 or Table 6.8, as appropriate (i.e. by the end of T_s or T_{hs}, respectively).

6.3.1.3.5 Minimum operating RF field strength and backscatter strength

For a Tag certified to this protocol, the Tag manufacturer shall specify:

1. free-space sensitivity,
2. minimum backscattered modulated power (ASK modulation) or change in radar cross-section or equivalent (phase modulation), and
3. the manufacturer's normal operating conditions,

for the Tag mounted on one or more manufacturer-selected materials.

6.3.1.4 Transmission order

The transmission order for all R=>T and T=>R communications shall be most-significant bit (MSB) first.

Within each message, the most-significant word shall be transmitted first.

Within each word, the MSB shall be transmitted first.

6.3.1.5 Cyclic-redundancy check (CRC)

A CRC is a cyclic-redundancy check that a Tag uses to ensure the validity of certain R=>T commands, and an Interrogator uses to ensure the validity of certain backscattered T=>R replies. This protocol uses two CRC types: (i) a CRC-16, and (ii) a CRC-5. Annex F describes both CRC types.

To generate a CRC-16 a Tag or Interrogator shall first generate the CRC-16 precursor shown in Table 6.11, and then take the ones-complement of the generated precursor to form the CRC-16.

Table 6.11 — CRC-16 precursor

CRC-16 precursor (See also Annex F)				
CRC Type	**Length**	**Polynomial**	**Preset**	**Residue**
ISO/IEC 13239	16 bits	$x^{16} + x^{12} + x^5 + 1$	$FFFF_h$	$1D0F_h$

A Tag or Interrogator shall verify the integrity of a received message that uses a CRC-16. The Tag or Interrogator may use one of the methods described in Annex F to verify the CRC-16.

A Tag calculates and saves into memory a 16-bit StoredCRC. See 6.3.2.1.2.1.

During inventory a Tag backscatters a 16-bit PacketCRC that the Tag calculates dynamically.

Tags shall append a CRC-16 to those replies that use a CRC-16. See 6.3.2.12 for command-specific replies.

Table 6.12 — CRC-5 definition

CRC-5 Definition (See also Annex F)				
CRC Type	**Length**	**Polynomial**	**Preset**	**Residue**
—	5 bits	$x^5 + x^3 + 1$	01001_2	00000_2

To generate a CRC-5 an Interrogator shall use the definition in Table 6.12.

A Tag shall verify the integrity of a received message that uses a CRC-5. The Tag may use the method described in Annex F to verify a CRC-5.

Interrogators shall append the appropriate CRC to R=>T transmissions as specified in Table 6.28.

6.3.1.6 Link timing

Figure 6.18 illustrates R=>T and T=>R link timing. The figure (not drawn to scale) defines Interrogator interactions with a Tag population. Table 6.16 shows the timing requirements for Figure 6.18, while 6.3.2.12 describes the commands. Tags and Interrogators shall meet all timing requirements shown in Table 6.16. RTcal is defined in 6.3.1.2.8; T_{pri} is the T=>R link period (T_{pri} = 1 / BLF). As described in 6.3.1.2.8, an Interrogator shall use a fixed R=>T link rate for the duration of an inventory round; prior to changing the R=>T link rate an Interrogator shall transmit CW for a minimum of 8 RTcal. Figure 6.18 illustrates three types of Tag reply timing denoted *immediate*, *delayed*, and *in-process*. These reply timings are defined in 6.3.1.6.1, 6.3.1.6.2, and 6.3.1.6.3, respectively. Table 6.28 specifies the reply type that a Tag uses for each Interrogator command.

6.3.1.6.1 Immediate Tag reply

An *immediate* Tag reply is a reply that meets T_1 specified in Table 6.16.

6.3.1.6.2 Delayed Tag reply

A *delayed* Tag reply is a reply that meets T_5 specified in Table 6.16. After issuing a command that uses *delayed* reply timing an Interrogator shall transmit CW for at least the lesser of T_{REPLY} or $T_{5(max)}$, where T_{REPLY} is the time between the Interrogator's command and the Tag's backscattered reply. An

Interrogator may observe several possible outcomes from a command that uses *delayed* reply timing, depending on the success or failure of the Tag's internal operations:

(a) **The Tag successfully executes the command:** After executing the command the Tag shall backscatter the reply shown in Table 6.13 and Figure 6.16, comprising a header (a 0-bit), the Tag's handle, and a CRC-16 calculated over the 0-bit and handle. The reply shall meet the T_5 limits in Table 6.16. If the Interrogator observes this reply within $T_{5(max)}$ then the command completed successfully.

(b) **The Tag encounters an error:** The Tag shall backscatter an error code (see Annex I) during the CW period rather than the reply shown in Table 6.13. The error code shall meet the T_5 limits in Table 6.16. An error-code reply uses header=1 (see Annex I) to differentiate it from a successful reply.

(c) **The Tag fails to execute the command:** If the Interrogator does not observe a Tag reply within $T_{5(max)}$ then the Tag did not execute the command successfully. The Interrogator typically issues a subsequent *Req_RN* (containing the Tag's handle) to verify that the Tag is still in the Interrogator's energizing RF field, and may then issue another command or commands.

A Tag shall ignore Interrogator commands while processing a prior command that specified a *delayed* reply. If an Interrogator transmits a command while the Tag is processing then the Tag may continue with its processing or, in environments with limited power availability, may undergo a power-on reset.

Table 6.13 — Tag reply after successfully executing a delayed-reply command

	Header	RN	CRC
# of bits	1	16	16
description	0	handle	CRC-16

A *delayed* Tag reply shall use the extended preamble shown in Figure 6.11 or Figure 6.15, as appropriate (i.e. the Tag shall reply as if TRext=1 regardless of the TRext value in the *Query* that initiated the inventory round).

Figure 6.16 — Successful delayed-reply sequence

6.3.1.6.3 In-process Tag reply

An *in-process* reply allows a Tag to spend longer than $T_{5(max)}$ executing a command, and to notify the Interrogator on a periodic basis that it is still processing the command. An *in-process* reply may include multiple backscatter transmissions from Tag to Interrogator. The first transmission shall meet the T_6 limits specified in Table 6.16; subsequent transmissions (if any) shall meet T_7. A Tag shall backscatter a transmission at least once every $T_{7(max)}$ while processing the command. A Tag may backscatter a "processing" notification more frequently than $T_{7(max)}$, and may backscatter an intermediate "processing" notification even if it can complete its processing within $T_{6(max)}$. An Interrogator specifies, in every access command that uses an *in-process* reply (except *AuthComm* – see 6.3.2.12.3.11), whether a Tag, when done processing, backscatters its final response or stores it in the Tag's ResponseBuffer. A Tag always backscatters (and never stores) the response to an *AuthComm*.

A Tag's *in-process* reply or replies shall be as shown in Table 6.14. The reply includes a 7-bit Barker code, done, header, optional length (length of the response regardless of whether the Tag backscatters or stores it), response (null if a Tag stores its response), the Tag's handle, and a CRC-16 calculated from Barker code to handle, inclusive. The Tag replies shall be consistent for first and subsequent Tag transmissions – i.e. if the final reply includes length then all intermediate replies shall include length, and vice versa.

An Interrogator may observe several possible outcomes from a command that uses *in-process* reply timing, depending on the success or failure of the Tag's internal operations:

(a) **The Tag successfully executes the command:** While processing the command the Tag backscatters a transmission as shown in Table 6.14 at least once every $T_{7(max)}$. Done and header for these intermediate replies shall be zero, response shall be null, and if the replies include length then length=0000_h. When the Tag has finished processing it sends a final reply, also as shown in Table 6.14, including done=1, header=0, response, and optional length. All replies shall meet the T_6 and T_7 limits specified in Table 6.16. If the Interrogator observes a final reply with header=0 then the command completed successfully.

(b) **The Tag encounters an error:** While processing the command the Tag backscatters a transmission as shown in Table 6.14 at least once every $T_{7(max)}$. Done and header for these intermediate replies shall be zero, response shall be null, and if the replies include length then length=0000_h. When a Tag encounters an error it sends a final reply, also as shown in Table 6.14, including done=1, header=1, response, and optional length. All replies shall meet the T_6 and T_7 limits specified in Table 6.16. If the Interrogator observes a final reply with header=1 then the Tag encountered an error (see Annex I).

(c) **The Tag fails to execute the command:** If the Interrogator does not observe a Tag reply within $T_{6(max)}$ for the first reply or $T_{7(max)}$ for subsequent replies then the Tag did not execute the command successfully. The Interrogator typically issues a subsequent *Req_RN* (containing the Tag's handle) to verify that the Tag is still in the Interrogator's energizing RF field, and may then issue another command or commands.

Done indicates whether the Tag is still processing a command. Done=0 means the Tag is still processing; done=1 means the Tag has finished processing. Header indicates whether response is an error code. Header=0 means the computation was successful and response includes a result; header=1 means response is an error code.

The optional length field specifies the length of a Tag's computed response (in bits), regardless of whether a Tag backscatters or stores this response. Length shall comprise a 15-bit value field followed by an even parity bit (the number of 1's in the 16-bit length field shall be an even number). An Interrogator specifies, in every command that uses an *in-process* reply, whether the Tag omits or includes length in its reply. If the Interrogator does not request length then the Tag omits length from its reply; if the Interrogator requests length then the Tag includes length in its reply. For the latter case, in the event of a stored response, length specifies the length of the stored response (and is therefore typically nonzero) but the response that the Tag actually backscatters will be null.

Response contains the Tag's computed result or an error code if the Interrogator instructed the Tag to backscatter its response, or null if the Interrogator instructed the Tag to store its response in the ResponseBuffer.

Table 6.15 shows values for the fields in an *in-process* reply depending on (a) whether a Tag sends its response or stores it in its ResponseBuffer, (b) how quickly the Tag executes the command, (c) whether the reply includes length, and (d) whether the Tag was able to successfully execute the command.

A Tag shall ignore Interrogator commands while processing a prior command that specified an *in-process* reply. If an Interrogator transmits a command while the Tag is processing then the Tag may continue with its processing or, in environments with limited power availability, may undergo a power-on reset.

After issuing a command that uses an *in-process* reply an Interrogator shall transmit CW until the Interrogator either (a) observes a reply with done=1 indicating the Tag has finished executing the command, or (b) fails to observe a reply for at least $T_{6(max)}$ or $T_{7(max)}$ (as appropriate) indicating that the Tag failed to execute the command.

An *in-process* Tag reply shall use the extended preamble shown in Figure 6.11 or Figure 6.15, as appropriate (i.e. the Tag shall reply as if TRext=1 regardless of the TRext value in the *Query* that initiated the inventory round).

6.3.1.6.4 ResponseBuffer

A Tag that implements a *Challenge* command, or any access command (other than *AuthComm*) that employs an *in-process* reply, shall implement a **C** flag and a ResponseBuffer with the following properties:

— The **C** flag, located in the Tag's XPC_W1 (see Table 6.18), indicates whether the Tag has a stored response (result or error code) in its ResponseBuffer. If **C**=1 then the ResponseBuffer contains data; if **C**=0 then the ResponseBuffer is empty.

— A Tag shall set **C**=0 upon receiving either an access command with SenRep=0 (c.f. 6.3.2.12.3.10) or a *Challenge* command, and shall set **C**=1 after finishing its processing and storing its response (result or error code) in its ResponseBuffer.

— The **C** flag shall be selectable using a *Select* command.

— If an access command with SenRep=0 or a *Challenge* command specifies IncRepLen=0 then a Tag shall not include a length field with its stored response, so the first word of the stored response shall be at ResponseBuffer location 00_h. If the command specifies IncRepLen=1 then ResponseBuffer bits 00_h – $0E_h$ shall contain the length of the stored response in bits, ResponseBuffer bit $0F_h$ shall contain an even parity bit that the Tag computes over bits 00_h – $0E_h$, and the first word of the stored response shall be at ResponseBuffer location 10_h. See Figure 6.17.

— The maximum size of a stored response shall be 32 kbits.

Table 6.14 — In-process Tag reply omitting and including length field

	Barker Code	Done	Header	Response	RN	CRC
# of bits	7	1	1	variable	16	16
description	1110010	0: Working 1: Finished	0: Success 1: Error code	result or error code	handle	CRC-16

	Barker Code	Done	Header	Length	Response	RN	CRC
# of bits	7	1	1	16	variable	16	16
description	1110010	0: Working 1: Finished	0: Success 1: Error code	15 bits encoding length of result or error code followed by even parity bit	result or error code	handle	CRC-16

— The maximum ResponseBuffer size shall be 32,784 bits (15 length bits, 1 parity bit, 32k response bits). A Tag manufacturer may limit the ResponseBuffer to a size less than this maximum. A Tag shall dynamically adjust its ResponseBuffer, on a command-by-command basis, to the required size.

— An Interrogator may read the ResponseBuffer using a *ReadBuffer* command. See 6.3.2.12.3.15.

— The ResponseBuffer shall be read-only to an Interrogator.

— A Tag shall abort command processing and instead store an error code in its ResponseBuffer if and when it determines that response will overflow the ResponseBuffer (see Annex I).

— A Tag shall retain data in its ResponseBuffer with the persistence of its **C** flag (see Table 6.20). When **C** is 1 then a Tag shall maintain the data in its ResponseBuffer. When **C** is or becomes 0 then a Tag shall deallocate its ResponseBuffer.

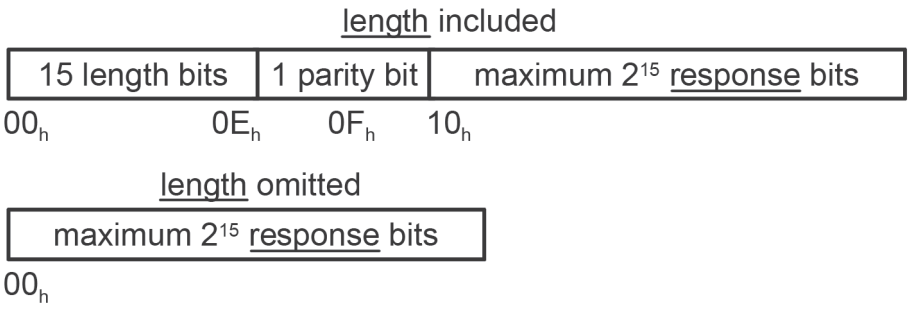


Figure 6.17 — ResponseBuffer data storage

This protocol does not specify a Tag memory location for the ResponseBuffer. Also because a ResponseBuffer is not writeable by an Interrogator, this protocol does not specify a mechanism for an Interrogator to write to it.

Table 6.15 — Possible in-process Tag replies

Sent or stored reply	Time for Tag to execute command?	Omit or include length?	Reply	
			Tag executed command	Tag failed to execute command
Sent	≤ $T_{6(max)}$	Omit	Done: 1 Header: 0 Response: result	Done: 1 Header: 1 Response: error code
		Include	Done: 1 Header: 0 Length: length of sent result Response: result	Done: 1 Header: 1 Length: length of sent error code Response: error code
	Any	Omit	Intermediate reply or replies Done: 0 Header: 0 Response: null Final reply Done: 1 Header: 0 Response: result	Intermediate reply or replies Done: 0 Header: 0 Response: null Final reply Done: 1 Header: 1 Response: error code
		Include	Intermediate reply or replies Done: 0 Header: 0 Length: 0000_h Response: null Final reply Done: 1 Header: 0 Length: length of sent result Response: result	Intermediate reply or replies Done: 0 Header: 0 Length: 0000_h Response: null Final reply Done: 1 Header: 1 Length: length of sent error code Response: error code

Table 6.15 *(continued)*

Sent or stored reply	Time for Tag to execute command?	Omit or include length?	Reply	
			Tag executed command	**Tag failed to execute command**
Stored	≤ $T_{6(max)}$	Omit	Done: 1 Header: 0 Response: null	Done: 1 Header: 1 Response: null
		Include	Done: 1 Header: 0 Length: length of stored result Response: null	Done: 1 Header: 1 Length: length of stored error code Response: null
	Any	Omit	Intermediate reply or replies Done: 0 Header: 0 Response: null Final reply Done: 1 Header: 0 Response: null	Intermediate reply or replies Done: 0 Header: 0 Response: null Final reply Done: 1 Header: 1 Response: null
		Include	Intermediate reply or replies Done: 0 Header: 0 Length: 0000_h Response: null Final reply Done: 1 Header: 0 Length: length of stored result Response: null	Intermediate reply or replies Done: 0 Header: 0 Length: 0000_h Response: null Final reply Done: 1 Header: 1 Length: length of stored error code Response: null

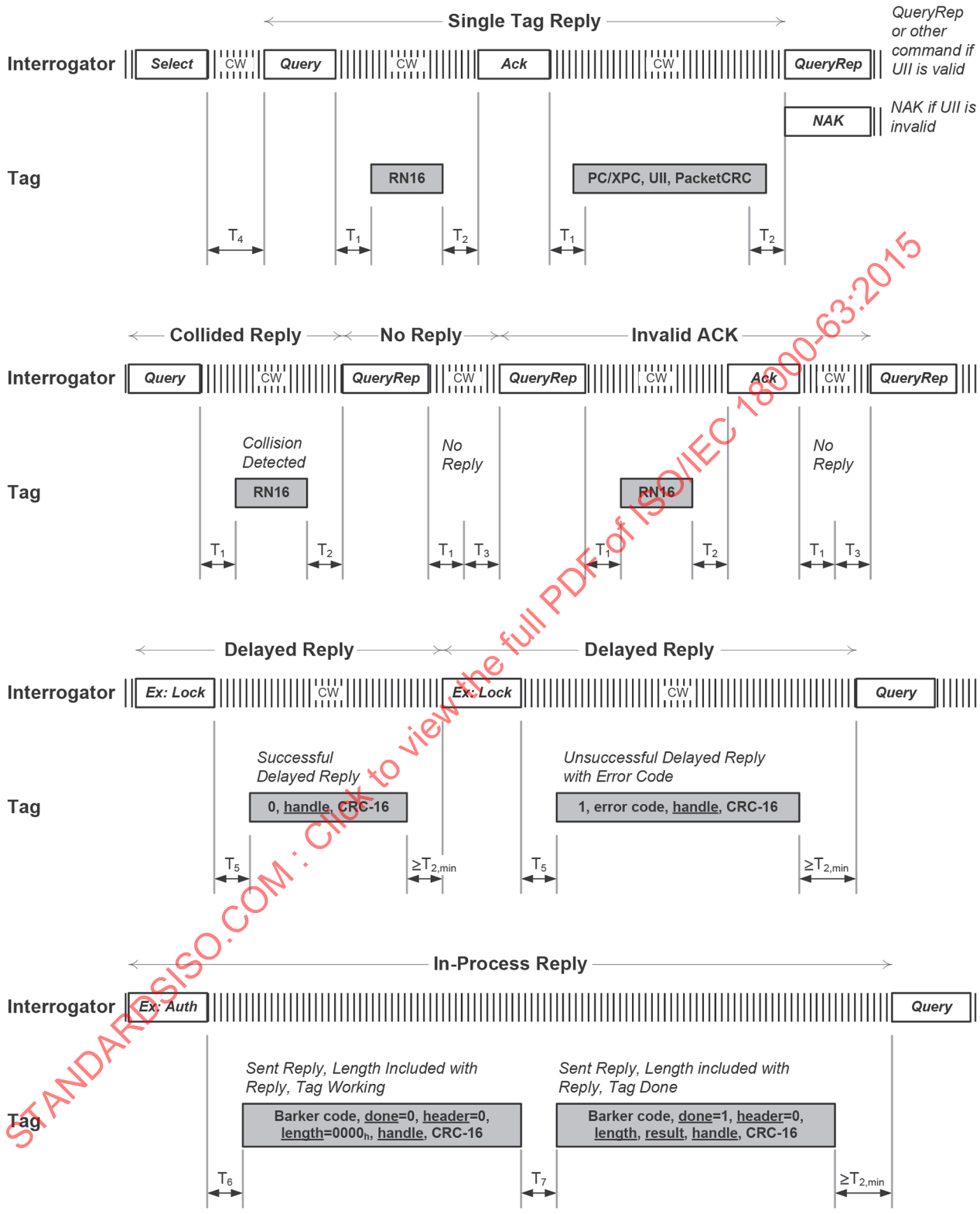


Figure 6.18 — Link timing

Table 6.16 — Link timing parameters

Parameter	Minimum	Nominal	Maximum	Description
T_1	MAX(RTcal,10T_{pri}) × (1 – |FrT|) – 2µs	MAX(RTcal,10T_{pri})	MAX(RTcal,10T_{pri}) × (1 + |FrT|) + 2µs	*Immediate* reply time from Interrogator transmission to Tag reply. Specifically, the time from the last rising edge of the last bit of the Interrogator transmission to the first rising edge of the Tag reply for an *immediate* Tag reply, measured at the Tag's antenna terminals.
T_2	3.0T_{pri}		20.0T_{pri}	Interrogator reply time if a Tag is to demodulate the Interrogator signal, measured from the end of the last (dummy) bit of the Tag reply to the first falling edge of the Interrogator transmission
T_3	0.0T_{pri}			Time an Interrogator waits, after T_1, before it issues another command
T_4	2.0 RTcal			Minimum time between Interrogator commands
T_5	MAX(RTcal,10T_{pri}) × (1 – |FrT|) – 2µs		20ms	*Delayed* reply time from Interrogator transmission to Tag reply. Specifically, the time from the last rising edge of the last bit of the Interrogator transmission to the first rising edge of the Tag reply for a *delayed* Tag reply, measured at the Tag's antenna terminals.
T_6	MAX(RTcal,10T_{pri}) × (1 – |FrT|) – 2µs		20ms	*In-process* reply time from Interrogator transmission to the first Tag reply. Specifically, the time from the last rising edge of the last bit of the Interrogator transmission to the first rising edge of the first Tag reply indicating that the Tag is either (a) still working, or (b) is done, measured at the Tag's antenna terminals
T_7	MAX(250µs, $T_{2(max)}$)		20ms	*In-process* reply time between Tag replies. Specifically, the time from the end of the last (dummy) bit of the Tag's prior transmission indicating that the Tag is still working to the first rising edge of the current Tag reply indicating that the Tag is either (a) still working, or (b) is done, measured at the Tag's antenna terminals

The following items apply to the requirements specified in Table 6.16:

1. T_{pri} denotes either the commanded period of an FM0 symbol or the commanded period of a single subcarrier cycle, as appropriate.

2. The maximum value for T_2 shall apply only to Tags in the **reply** or **acknowledged** states (see 6.3.2.6.3 and 6.3.2.6.4). For a Tag in the **reply** or **acknowledged** states, if T_2 expires (i.e. reaches its maximum value):

   — Without the Tag receiving a valid command, the Tag shall transition to the **arbitrate** state (see 6.3.2.6.2),

   — During the reception of a valid command, the Tag shall execute the command,

   — During the reception of an invalid command, the Tag shall transition to **arbitrate** upon determining that the command is invalid.

In all other states the maximum value for T_2 shall be unrestricted. A Tag shall be allowed a tolerance of $20.0T_{pri} \leq T_{2(max)} \leq 32T_{pri}$ in determining whether T_2 has expired. "Invalid command" is defined in 6.3.2.12.

3. An Interrogator may transmit a new command prior to interval T_2 (i.e. during a Tag response). In this case the responding Tag may ignore the new command and, in environments with limited power availability, and may undergo a power-on reset.

4. FrT is the frequency tolerance specified in Table 6.9.

5. T_1+T_3 shall not be less than T_4.

6.3.2 Logical interface

The logical interface between an Interrogator and a Tag may be viewed as the lowest level in the data link layer of a layered network communication system. The logical interface defines Tag memory, flags, states, selection, inventory, and access.

6.3.2.1 Tag memory

Tag memory shall be logically separated into the four distinct memory banks shown in Figure 6.19, each of which may comprise zero or more memory words. The memory banks are:

Reserved memory shall contain the kill and and/or access passwords, if passwords are implemented on the Tag. The kill password shall be stored at memory addresses 00_h to $1F_h$; the access password shall be stored at memory addresses 20_h to $3F_h$. See 6.3.2.1.1.

UII memory shall contain a StoredCRC at memory addresses 00_h to $0F_h$, a StoredPC at addresses 10_h to $1F_h$, a code (such as a UII, and hereafter referred to as a UII) that identifies the object to which the Tag is or will be attached beginning at address 20_h, and if the Tag implements Extended Protocol Control (XPC) then either one or two XPC word(s) beginning at address 210_h. See 6.3.2.1.2.

TID memory shall contain an 8-bit ISO/IEC 15963 allocation class identifier at memory locations 00_h to 07_h. TID memory shall contain sufficient identifying information above 07_h for an Interrogator to uniquely identify the custom commands and/or optional features that a Tag supports. See 6.3.2.1.3.

User memory is optional. If a Tag implements User memory then it may partition the User memory into one or more files. If the Tag implements a single file then that file is File_0. See 6.3.2.1.4 and 6.3.2.11.3.

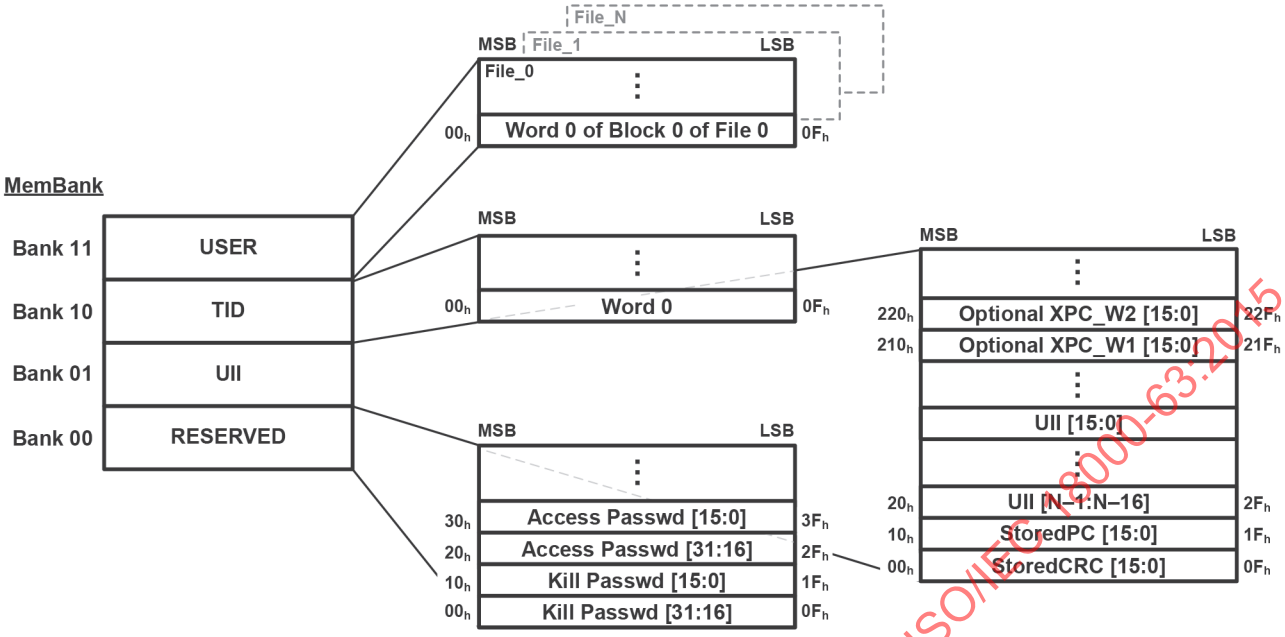


Figure 6.19 — Logical memory map

The logical addressing of all memory banks and User-memory files shall begin at 00_h. The physical memory map is Tag-manufacturer defined. When a Tag backscatters data from memory the order is left-to-right and bottom-to-top in Figure 6.19. The backscatter shall fall on word boundaries (except for a truncated reply – see 6.3.2.12.1.1).

MemBank shall be defined as follows:

00_2	Reserved	01_2	UII
10_2	TID	11_2	User

Operations in one logical memory bank shall not access memory locations in another bank.

Memory writes involve the transfer of one or more 16-bit words from Interrogator to Tag. A *Write* command writes 16 bits (i.e. one word) at a time, using link cover-coding to obscure the data during R=>T transmission. The optional *BlockWrite* command writes one or more 16-bit words at a time, without link cover-coding. The optional *BlockErase* command erases one or more 16-bit words at a time. A *Write*, *BlockWrite*, or *BlockErase* shall not alter a Tag's killed status regardless of the memory address (whether valid or invalid) specified in the command.

An Interrogator may issue a *Lock* command (see 6.3.2.12.3.5) to lock, permanently lock, unlock, or permanently unlock the kill password, access password, UII memory bank, TID memory bank, or File_0 of User memory, thereby preventing or allowing subsequent changes (as appropriate). If the passwords are locked or permanently locked then they are unwriteable and unreadable by any command and usable only by a *Kill* or *Access* command. If UII memory, TID memory, or File_0 are locked or permanently locked then they are unwriteable but readable, except for the **L** and **U** bits in UII memory; an Interrogator with an asserted Untraceable privilege may alter the **L** and **U** bits regardless of the lock or permalock status of UII memory (see 6.3.2.12.3.16).

If a Tag implements User memory then it partitions each File_N, N≥0 of User memory into one or more equal-size blocks. A Tag shall use the same block size for file allocation (see 6.3.2.11.3) as it does for the *BlockPermalock* command (see 6.3.2.12.3.9). A Tag may use different block sizes for the *BlockWrite* and *BlockErase* commands. If a Tag supports the *BlockPermalock* command then an Interrogator may issue a *BlockPermalock* to permanently lock one or more memory blocks. If blocks within File_0 are permalocked then these blocks are permanently unwriteable but readable. If blocks within File_N, N>0

are permalocked then these blocks are permanently unwriteable but readable by an Interrogator with appropriate read privileges (see Table 6.24 and Table 6.25).

6.3.2.1.1 Reserved Memory

Reserved memory contains the kill (see 6.3.2.1.1.1) and/or access (see 6.3.2.1.1.2) passwords, if passwords are implemented on a Tag. If a Tag does not implement the kill and/or access password(s) then the Tag shall logically operate as though it has zero-valued password(s) that are permanently read/write locked (see 6.3.2.12.3.5), and the corresponding physical memory locations in Reserved memory need not exist.

6.3.2.1.1.1 Kill password

The kill password is a 32-bit value stored in Reserved memory 00_h to $1F_h$, MSB first. The default (unprogrammed) value shall be zero. A Tag that does not implement a kill password shall behave as though it has a zero-valued kill password that is permanently read/write locked. A Tag shall not execute a password-based kill if its kill password is zero (see 6.3.2.12.3.4). An Interrogator may use a nonzero kill password in a password-based *Kill*-command sequence to kill a Tag and render it nonresponsive thereafter.

6.3.2.1.1.2 Access password

The access password is a 32-bit value stored in Reserved memory 20_h to $3F_h$, MSB first. The default (unprogrammed) value shall be zero. A Tag that does not implement an access password shall behave as though it has a zero-valued access password that is permanently read/write locked. A Tag with a zero-valued access password transitions from the **acknowledged** state to the **secured** state upon commencing access, without first entering the **open** state. A Tag with a nonzero-valued access password transitions from the **acknowledged** state to the **open** state upon commencing access; an Interrogator may then use the access password in an *Access* command sequence to transition the Tag from the **open** to the **secured** state.

6.3.2.1.2 UII Memory

UII memory contains a StoredCRC at addresses 00_h to $0F_h$, a StoredPC at 10_h to $1F_h$, a UII beginning at 20_h, and optional first and second XPC words at 210_h – $21F_h$ (XPC_W1) and 220_h – $22F_h$ (XPC_W2), respectively. The StoredCRC, StoredPC, UII, and XPC word(s) shall be stored MSB first (i.e. the UII's MSB is at location 20_h).

The StoredCRC and StoredPC are described in 6.3.2.1.2.1 and 6.3.2.1.2.2, respectively.

The UII identifies the object to which the Tag is affixed. The UII for GS1 EPCglobal™ Applications is described in 6.3.2.1.2.3; the UII for ISO Applications is described in 6.3.2.1.2.4. An Interrogator may issue a *Select* that includes all or part of the UII in its Mask. An Interrogator may issue an *ACK* to cause a Tag to backscatter its UII. Under certain circumstances a Tag may truncate its backscattered UII (see 6.3.2.12.3.16 and 6.3.2.12.1.1). An Interrogator may issue a *Read* to read all or part of the UII.

The XPC_W1 and XPC_W2 are described in 6.3.2.1.2.5.

6.3.2.1.2.1 CRC-16 (StoredCRC and PacketCRC)

A Tag shall implement both a StoredCRC and a PacketCRC. The StoredCRC is stored in UII memory, is selectable by an Interrogator using a *Select* command, and is readable by an Interrogator using a *Read* command. The PacketCRC is computed and sent by a Tag during backscatter, protects the transmitted PC/XPC and UII, and is neither selectable nor directly readable by an Interrogator.

A Tag shall compute and store its StoredCRC either (i) when an Interrogator writes or overwrites bits in the UII (including in the StoredPC), or (ii) every time the Tag powers up. The Tag manufacturer shall choose whether the Tag implements (i) or (ii). A Tag shall perform its computing and storing for these two cases as follows:

(i) The Tag first writes or overwrites the bits, then computes and stores a new StoredCRC, all within the reply times specified in Table 6.16 for the command (*Write*, *BlockWrite*, *BlockErase*, or *Untraceable*) that wrote or overwrote the bits. A Tag shall delay backscattering the success reply shown in Table 6.13 or Table 6.14 for the command that wrote or overwrote the bits until it has stored the new StoredCRC. The Tag shall store its StoredCRC in nonvolatile memory so that the StoredCRC persists through subsequent Tag power cycles.

(ii) The Tag computes and stores the StoredCRC before the end of interval T_s or T_{hs} (as appropriate) in Figure 6.3 or Figure 6.5, respectively. The Tag may store its StoredCRC in volatile or nonvolatile memory. If an Interrogator modifies a Tag's StoredPC or UII after Tag powerup then the StoredCRC may be incorrect until the Interrogator power-cycles the Tag.

For both cases (i) and (ii) the Tag shall implement the StoredCRC by first calculating a CRC-16 (see 6.3.1.5) over the StoredPC and the UII specified by the length (**L**) bits in the StoredPC, and then storing the thus-computed StoredCRC into UII memory 00_h to $0F_h$, MSB first. The Tag shall calculate the StoredCRC on word boundaries, shall deassert all Tag-computed StoredPC bit values (**XI** and **UMI** if Tag-computed) when performing the calculation, and shall omit XPC_W1 and XPC_W2 from the calculation.

If an Interrogator attempts to write to UII memory 00_h – $0F_h$ then the Tag shall not execute the write and instead treat the command's parameters as unsupported (see Table C.30).

In response to an *ACK* a Tag backscatters a reply comprising a PC word, in some instances an XPC word or words, the UII (which may be truncated), and a PacketCRC to protect the backscattered data stream (see Table 6.17). A Tag shall compute the PacketCRC as specified in 6.3.1.5 over the PC word, optional XPC word(s), and backscattered UII, and shall send the PacketCRC MSB first.

As required by 6.3.1.5 an Interrogator shall verify the integrity of the received PC word, optional XPC word or words, and UII using the PacketCRC.

In some circumstances the PacketCRC will differ from a Tag's StoredCRC, such as, for example, if the Tag has an asserted **XI** or the UII is truncated.

6.3.2.1.2.2 Protocol-control (PC) word (StoredPC and PacketPC)

A Tag shall implement a StoredPC in addresses 10_h–$1F_h$ of UII memory. The bit assignments for this StoredPC shall be as shown in Table 6.18 and defined in Table 6.19. Note that some bit assignments are different for GS1 EPCglobal (**T**=0) versus ISO (**T**=1) Applications. Similarly, some bit assignments for XPC_W1 differ with the Application (see 6.3.2.1.2.5), as does the method of computing **XI** (see below).

The StoredPC bits and values shall be as follows:

— **L (UII length field, bits 10_h – 14_h):** Bits 10_h – 14_h are written by an Interrogator and specify the length of the UII that a Tag backscatters in response to an *ACK*, in words:

  — 00000_2: Zero words.

  — 00001_2: One word (addresses 20_h to $2F_h$ in UII memory).

  — 00010_2: Two words (addresses 20_h to $3F_h$ in UII memory).

  •

  •

  •

  — 11111_2: 31 words (addresses 20_h to $20F_h$ in UII memory).

If a Tag only supports **XI**=0 then the maximum value for the UII length field in the StoredPC shall be 11111_2 (allows a 496-bit UII), as shown above. If a Tag supports **XI**=1 then the maximum value for the UII length field in the StoredPC shall be 11101_2 (allows a 464-bit UII). A Tag that supports **XI**=1 shall not execute a *Write*, *BlockWrite*, or *Untraceable* that attempts to write a UII length field larger than 11101_2 and shall instead treat the command's parameters as unsupported (see Table C.30).

— **UMI (User-memory indicator, bit 15_h):** Bit 15_h may be fixed by the Tag manufacturer or computed by the Tag. In the former (fixed) case, if the Tag does not have and is incapable of allocating memory to File_0 then the Tag manufacturer shall set bit 15_h to 0_2; if the Tag has or is capable of allocating memory to File_0 then the Tag manufacturer shall set bit 15_h to 1_2. In the latter (computed) case, both at power-up and upon writing the first word (bits 00_h – $0F_h$) of File_0 a Tag shall compute the logical OR of bits 03_h – 07_h of File_0 and shall map the computed value into bit 15_h; if the Tag does not have memory allocated to File_0 then the logical OR result shall be 0_2. Regardless of the **UMI** method (fixed or computed), when an Interrogator writes the StoredPC the Tag shall not write and instead ignore the data value the Interrogator provides for bit 15_h. For a computed **UMI**, if an Interrogator deallocates File_0 (see 6.3.2.11.3) then the Tag shall set bit 15_h to 0_2 upon deallocation. Also for a computed **UMI**, the untraceability status of User memory (see 6.3.2.11.1) shall not change the **UMI** value (i.e. if **UMI**=1 when a Tag is traceable then **UMI** shall remain 1 even if an Interrogator instructs a Tag to untraceably hide User memory).

— **XI (XPC_W1 indicator, bit 16_h):** If a Tag does not implement XPC_W1 then bit 16_h shall be fixed at 0_2 by the Tag manufacturer. If a Tag implements XPC_W1 then a Tag shall compute **XI** both at powerup and upon changing any bits of XPC_W1 (whether these bits are written or computed) and map the computed value into bit 16_h as follows: If **T**=0 then **XI** may be either (i) the logical OR of bits 210_h–217_h of UII memory or (ii) the logical OR of bits 210_h–218_h of UII memory; the Tag manufacturer shall choose whether the Tag implements (i) or (ii). If **T**=1 then **XI** is the logical OR of bits 210_h–$21F_h$ of UII memory. Regardless of whether **XI** is fixed or computed, when an Interrogator writes the StoredPC the Tag shall not write and instead ignore the data value the Interrogator provides for bit 16_h.

— **T (numbering system identifier toggle, bit 17_h):** If bit 17_h is 0_2 then the application is referred to as a GS1 EPCglobal™ Application and PC bits 18_h – $1F_h$ shall be as defined in this protocol. If bit 17_h is 1_2 then the application is referred to as a ISO Application and bits 18_h – $1F_h$ shall be as defined in ISO/IEC 15961.

— **RFU or AFI (Reserved for Future Use or Application Family Identifier, bits 18_h – $1F_h$):** If **T**=0 then the Tag manufacturer (if the bits are not writeable) or an Interrogator (if the bits are writeable) shall set these bits to 00_h. If **T**=1 then the Tag manufacturer (if the bits are not writeable) or an Interrogator (if the bits are writeable) shall set these bits as specified in ISO/IEC 15961.

If an Interrogator changes the UII length (via a memory write operation), and if it wishes the Tag to subsequently backscatter the new UII length, then it must write new **L** bits into the Tag's StoredPC. If an Interrogator attempts to write **L** bit values that the Tag does not support then the Tag shall not execute the write operation and instead treat the command's parameters as unsupported (see Table C.30).

A Tag that supports **XI**=1 shall implement a PacketPC in addition to a StoredPC. Which PC word a Tag backscatters in reply to an *ACK* shall be as defined in Table 6.17. A PacketPC differs from a StoredPC in its **L** bits, which a Tag adjusts to match the length of the backscattered data that follow the PC word. Specifically, if **XI**=1 but **XEB**=0 then a Tag backscatters an XPC_W1 before the UII, so the Tag shall add one to (i.e. increment) its **L** bits. If both **XI**=1 and **XEB**=1 then the Tag backscatters both an XPC_W1 and an XPC_W2 before the UII, so the Tag shall add two to (i.e. double increments) its **L** bits. Because Tags that support XPC functionality have a maximum **L** value of 11101_2, double incrementing increases the value to 11111_2. A Tag shall not, under any circumstances, allow its **L** bits to roll over to 00000_2. Note that incrementing or double incrementing the **L** bits does not alter the bit values stored in UII memory 10_h – 14_h; rather, a Tag increments the **L** bits in the backscattered PacketPC but leaves the memory contents unaltered.

A Tag that does not implement an XPC_W1 or untraceability need not implement a PacketPC.

The fields that a Tag includes in its reply to an *ACK* (Table 6.17) depend on the values of **T**, **C**, **XI**, and **XEB** (see Table 6.19); whether the Tag implements an XPC_W1; whether the Tag is truncating its reply

(see 6.3.2.12.1.1); and the value of immed (see 6.3.2.12.1.2). If a Tag has **T**=0, **XI**=0, implements an XPC_W1, and is not truncating then the Tag substitutes the 8 LSBs of XPC_W1 (i.e. UII memory 218_h – $21F_h$) for the 8 LSBs of the StoredPC (i.e. PC memory 18_h – $1F_h$) in its reply. Because a Tag calculates its PacketCRC over the backscattered data bits (see 6.3.2.1.2.1), when the Tag does this substitution then it shall calculate its PacketCRC over the 8 substituted XPC_W1 LSBs rather than over the 8 StoredPC LSBs.

An Interrogator shall support Tag replies with **XI**=0, **XI** =1, or both **XI**=1 and **XEB**=1.

When sending a truncated UII a Tag substitutes 00000_2 for its PC field — see Table 6.17 and 6.3.2.12.1.1.

If a Tag has a response (result or error code) in its ResponseBuffer (i.e. **C**=1) and the Interrogator set immed=1 in the *Challenge* command that preceded the inventory round then a Tag shall concatenate response and a CRC-16 calculated over response to its reply to an *ACK* (see Table 6.17).

Table 6.17 — Tag reply to an *ACK* command

T	XI	XEB	Truncation	C AND immed	Tag Backscatter					
					PC	XPC	UII [1]	CRC	Response	CRC
0	0	0	0	0	It Tag does not implement XPC_W1: ⇒ StoredPC(10_h–$1F_h$) If Tag implements XPC_W1 (see Note 2) ⇒ StoredPC(10_h–17_h), XPC_W1(218_h–$21F_h$)	None	Full	PacketCRC	–	–
0	0	0	0	1	**C**=1 so Tag implements XPC_W1 (see Note 2): ⇒ StoredPC(10_h–17_h), XPC_W1(218_h–$21F_h$)	None	Full	PacketCRC	response	CRC-16
0	0	0	1	0	00000_2	None	Truncated	PacketCRC	–	–
0	0	0	1	1	00000_2	None	Truncated	PacketCRC	response	CRC-16
0	0	1	All		Disallowed [3]					
0	1	0	0	0	PacketPC	XPC_W1	Full	PacketCRC	–	–
0	1	0	0	1	PacketPC	XPC_W1	Full	PacketCRC	response	CRC-16
0	1	0	1	0	00000_2	None	Truncated	PacketCRC	–	–
0	1	0	1	1	00000_2	None	Truncated	PacketCRC	response	CRC-16
0	1	1	0	0	PacketPC	XPC_W1, XPC_W2	Full	PacketCRC	–	–
0	1	1	0	1	PacketPC	XPC_W1, XPC_W2	Full	PacketCRC	response	CRC-16
0	1	1	1	0	00000_2	None	Truncated	PacketCRC	–	–
0	1	1	1	1	00000_2	None	Truncated	PacketCRC	response	CRC-16
1	0	0	0	0 (**C**=0)	StoredPC(10_h–$1F_h$)	None	Full	PacketCRC	–	–
1	0	0	0	0 (**C**=1)	Disallowed [4]					
1	0	0	0	1	Disallowed [5]					
1	0	0	1	0 (**C**=0)	00000_2	None	Truncated	PacketCRC	–	–

Table 6.17 *(continued)*

T	XI	XEB	Truncation	C AND immed	Tag Backscatter					
					PC	XPC	UII [1]	CRC	Response	CRC
1	0	0	1	0 (C=1)	Disallowed [5]					
1	0	0	1	1	Disallowed [5]					
1	0	1	All		Disallowed [3]					
1	1	0	0	0	PacketPC	XPC_W1	Full	PacketCRC	–	–
1	1	0	0	1	PacketPC	XPC_W1	Full	PacketCRC	response	CRC-16
1	1	0	1	0	00000_2	None	Truncated	PacketCRC	–	–
1	1	0	1	1	00000_2	None	Truncated	PacketCRC	response	CRC-16
1	1	1	0	0	PacketPC	XPC_W1, XPC_W2	Full	PacketCRC	–	–
1	1	1	0	1	PacketPC	XPC_W1, XPC_W2	Full	PacketCRC	response	CRC-16
1	1	1	1	0	00000_2	None	Truncated	PacketCRC	–	–
1	1	1	1	1	00000_2	None	Truncated	PacketCRC	response	CRC-16

Note 1 Full means a UII whose length is specified by the L bits in the StoredPC; truncated means a UII whose length is shortened by a prior *Select* command specifying truncation (see 6.3.2.12.1.1).

Note 2 If a Tag has T=0, XI=0, implements an XPC_W1, and is not truncating then the Tag substitutes UII memory bits 218_h–$21F_h$ for UII memory bits 18_h–$1F_h$ in its reply to an *ACK*.

Note 3 If T=0 then XI may be either (i) the logical OR of bits 210_h–217_h of XPC_W1 or (ii) the logical OR of bits 210_h–218_h of XPC_W1; the Tag manufacturer chooses whether a Tag implements (i) or (ii). If T=1 then XI is the logical OR of the entirety of XPC_W1 (210_h–$21F_h$). Because XEB is the MSB (210_h) of XPC_W1, if XEB=1 then XI=1 regardless of the T value.

Note 4 If T=1 then XI is the logical OR of the entirety of XPC_W1 (210_h–$21F_h$), so if C=1 then XI=1.

Table 6.18 — StoredPC and XPC_W1 bit assignments

StoredPC bit assignments																
Application	MSB															LSB
	10_h	11_h	12_h	13_h	14_h	15_h	16_h	17_h	18_h	19_h	$1A_h$	$1B_h$	$1C_h$	$1D_h$	$1E_h$	$1F_h$
GS1 EPCglobal	**L4**	**L3**	**L2**	**L1**	**L0**	**UMI**	**XI**	**T**=0	RFU							
ISO	**L4**	**L3**	**L2**	**L1**	**L0**	**UMI**	**XI**	**T**=1	AFI as defined in ISO/IEC 15961							

XPC_W1 bit assignments																
Application	MSB															LSB
	210_h	211_h	212_h	213_h	214_h	215_h	216_h	217_h	218_h	219_h	$21A_h$	$21B_h$	$21C_h$	$21D_h$	$21E_h$	$21F_h$
GS1 EPCglobal	**XEB**	RFU							**B**	**C**	**SLI**	**TN**	**U**	**K**	**NR**	**H**
ISO	**XEB**	As defined in ISO/IEC 18000-63							**B**	**C**	**SLI**	**TN**	**U**	**K**	**NR**	**H**

Table 6.19 — StoredPC and XPC_W1 bit values

Hex	Name	How Set? [1]	Descriptor	Settings
10:14	**L4**:**L0**	Written	UII length field	**L4**:**L0** encode a numeric value. See 6.3.2.1.2.2.
15	**UMI**	Fixed or computed	File_0 indicator	0: If fixed then Tag does not have and is incapable of allocating memory to File_0. If computed then File_0 is not allocated or does not contain data. 1: If fixed then Tag has or is capable of allocating memory to File_0. If computed then File_0 is allocated and contains data.
16	**XI**	Computed	XPC_W1 indicator	0: Either (i) Tag has no XPC_W1, or (ii) **T**=0 and either bits 210_h–217_h or bits 210_h–218_h (at Tag-manufacturers option) of UII memory are all zero, or (iii) **T**=1 and bits 210_h–$21F_h$ of UII memory are all zero. 1: Tag has an XPC_W1 and either (i) **T**=0 and at least one bit of 210_h–217_h or 210_h–218_h (at Tag-manufacturer's option) of UII memory is nonzero, or (ii) **T**=1 and at least one bit of 210_h–$21F_h$ of UII memory is nonzero
17	**T**	Written	Numbering System Identifier Toggle	0: Tag is used in a GS1 EPCglobal™ Application 1: Tag is used in an ISO Application
18:1F	RFU or AFI	Per the Application	RFU or AFI	GS1 EPCglobal™ Application: RFU and fixed[1] at zero ISO Application: See ISO/IEC 15961
210	XEB	Computed	XPC_W2 indicator	0: Tag has no XPC_W2 or all bits of XPC_W2 are zero-valued 1: Tag has an XPC_W2 and at least one bit of XPC_W2 is nonzero
211	RFU	Per the Application	RFU or assigned by ISO/IEC 18000-63	GS1 EPCglobal™ Application: RFU and fixed[1] at zero ISO Application: RFU
212	RFU / MIIM	Per the Application	RFU or assigned by ISO/IEC 18000-63	GS1 EPCglobal™ Application: RFU and fixed[1] at zero ISO Application: Mobile RFID content name indicator (reserved for ISO/IEC 29143)
213	RFU / MIIM	Per the Application	RFU or assigned by ISO/IEC 18000-63	GS1 EPCglobal™ Application: RFU and fixed[1] at zero ISO Application: Reserved for ISO/IEC 29143 for future extensions
214	RFU / SA	Per the Application	RFU or assigned by ISO/IEC 18000-63	GS1 EPCglobal™ Application: RFU and fixed[1] at zero ISO Application: General Alarm
215	RFU / SS	Per the Application	RFU or assigned by ISO/IEC 18000-63	GS1 EPCglobal™ Application: RFU and fixed[1] at zero ISO Application: Simple Sensor
216	RFU / FS	Per the Application	RFU or assigned by ISO/IEC 18000-63	GS1 EPCglobal™ Application: RFU and fixed[1] at zero ISO Application: Full Sensor
217	RFU	Per the Application	RFU or assigned by ISO/IEC 18000-63	GS1 EPCglobal™ Application: RFU and fixed[1] at zero ISO Application: RFU
218	B	Tag mfg defined	Battery-Assisted Passive indicator	0: Tag is passive or does not support the **B** flag 1: Tag is battery-assisted

Table 6.19 *(continued)*

Hex	Name	How Set? [1]	Descriptor	Settings
219	C	Computed	Computed response indicator	0: ResponseBuffer is empty or Tag does not support a ResponseBuffer 1: ResponseBuffer contains a response
21A	SLI	Computed	**SL** indicator	0: Tag has a deasserted **SL** flag or does not support the **SLI** bit 1: Tag has an asserted **SL** flag
21B	TN	Tag mfg defined	Notification indicator	0: Tag does not assert a notification or does not support the **TN** bit 1: Tag asserts a notification
21C	U	Written	Untraceable indicator	0: Tag is traceable or does not support the **U** bit 1: Tag is untraceable
21D	K	Computed	Killable indicator	0: Tag is not killable by *Kill* command or does not support the **K** bit 1: Tag can be killed by *Kill* command.
21E	NR	Written	Nonremovable indicator	0: Tag is removable from its host item or does not support the **NR** bit 1: Tag is not removable from its host item
21F	H	Written	Hazmat indicator	0: Tagged item is not hazardous material or Tag does not support the **H** bit 1: Tagged item is hazardous material

Note 1 "Written" indicates that an Interrogator writes the value; "computed" indicates that a Tag computes the value; "fixed" indicates that the Tag manufacturer fixes the value; "Tag mfg defined" indicates that the Tag manufacturer defines the bit settability (written, computed, or fixed). Written bits inherit the lock/permalock status of the UII memory bank (note: An *Untraceable* command may alter **L** and/or **U** regardless of the lock/permalock status of the UII memory bank). Computed bits are not writeable and may change despite the lock/permalock status of the UII memory bank. Fixed bits are not writeable and not changeable in the field.

6.3.2.1.2.3 UII for a GS1 EPCglobal™ Application

The UII for an GS1 EPCglobal™ Application shall be as defined in the GS1 EPC Tag Data Standard.

6.3.2.1.2.4 UII for an ISO Application

The UII for an ISO Application shall be as defined in ISO/IEC 15962.

6.3.2.1.2.5 Extended Protocol Control (XPC) word or words (optional)

A Tag may implement an XPC_W1 logically located at addresses 210_h to $21F_h$ of UII memory. If a Tag implements an XPC_W1 then it may additionally implement an XPC_W2 logically located at address 220_h to $22F_h$ of UII memory. A Tag shall not implement an XPC_W2 without also implementing an XPC_W1. If implemented, these XPC words shall be exactly 16 bits in length and stored MSB first. If a Tag does not support one or both of these XPC words then the specified memory locations need not exist. When an Interrogator writes the XPC a Tag shall not write and shall instead ignore the data values the Interrogator provides for the **C**, **SLI**, and **K** flags.

A Tag shall not implement any non-XPC memory element at UII memory locations 210_h to $22F_h$, inclusive. This requirement shall apply both to Tags that support an XPC word or words and to those that do not.

If a Tag implements an XPC_W1 then the Tag shall compute **XI** as described in 6.3.2.1.2.2. If a Tag implements an XPC_W2 then the Tag shall compute **XEB** as the logical OR of bits 220_h to $22F_h$ of UII memory, inclusive. A Tag shall perform these calculations both at powerup and upon changing any

bits 220_h to $22F_h$ of UII memory. A Tag shall perform its **XEB** calculation prior to performing its **XI** calculation so that if **XEB**=1 then **XI**=1.

If a Tag computes a bit in XPC_W1 or XPC_W2 and, as a result of a commanded operation, the Tag alters the bit value then the Tag shall map the new value into memory prior to executing a subsequent command.

An Interrogator may issue a *Select* command (see 6.3.2.12.1) with a Mask covering all or part of XPC_W1 and/or XPC_W2. An Interrogator may read a Tag's XPC_W1 and XPC_W2 using a *Read* command (see 6.3.2.12.3.2).

The XPC_W1 bits and values shall be as follows (see also Table 6.18 and Table 6.19).

— **XEB (bit 210_h):** If bit 210_h is 0_2 then either a Tag has no XPC_W2 or all bits of XPC_W2 are zero-valued. If bit 210_h is 1_2 then a Tag has an XPC_W2 and at least one bit of XPC_W2 is nonzero.

— **RFU or As Defined in ISO/IEC 18000-63 (bits 211_h–217_h):** If **T**=0 then the Tag manufacturer (if the bits are not writeable) or an Interrogator (if the bits are writeable) shall set bits 211_h–217_h to zero. If **T**=1 then a Tag and/or an Interrogator shall set bits 211_h–217_h as defined in ISO/IEC 18000-63.

— **B (Battery-assisted passive indicator, bit 218_h):** If bit 218_h is 0_2 then a Tag is either passive or does not support the **B** flag. If bit 218_h is 1_2 then a Tag is battery assisted.

— **C (Computed response indicator, bit 219_h):** If bit 219_h is 0_2 then the Tag's ResponseBuffer is empty or the Tag does not support a ResponseBuffer. If bit 219_h is 1_2 then a Tag has a response in its ResponseBuffer. See 6.3.1.6.4.

— **SLI (SL-flag indicator, bit $21A_h$):** If bit $21A_h$ is 0_2 then a Tag has a deasserted **SL** flag or does not support an **SL** indicator. If bit $21A_h$ is 1_2 then a Tag has an asserted **SL** flag. Upon receiving a *Query* a Tag that implements the **SL** indicator shall map its **SL** flag into the **SLI** and shall retain this **SLI** setting until starting a subsequent inventory round.

— **TN (Tag-notification indicator, bit $21B_h$):** If bit $21B_h$ is 0_2 then a Tag does not have a Tag notification or does not support the **TN** flag. If bit $21B_h$ is 1_2 then a Tag has a Tag notification. The indication provided by the **TN** bit is Tag-manufacturer defined and not specified by this protocol. A Tag manufacturer may configure the **TN** bit to be writeable, computed, or fixed. Depending on the manufacturer's implementation the **TN** bit may or may not inherit the permalock status of the UII memory bank.

— **U (Untraceable indicator, bit $21C_h$):** If bit $21C_h$ is 0_2 then either (i) the Tag does not support the **U** bit or (ii) an Interrogator has not asserted the **U** bit. If bit $21C_h$ is 1_2 then an Interrogator has asserted the **U** bit, typically for the purpose of indicating that the Tag is persistently reducing its operating range and/or is untraceably hiding memory. See 6.3.2.12.3.16.

— **K (Killable indicator, bit $21D_h$):** If bit $21D_h$ is 0_2 then a Tag is not killable or does not support the **K** bit. If bit $21D_h$ is 1_2 then a Tag is killable. Logically, **K** is defined as:

  ***K** = [(logical OR of AuthKill privilege for all keys) OR (logical OR of all 32 bits of the kill password) OR (kill-pwd-read/write=0) OR (kill-pwd-permalock=0)].* See also Table 6.21. In words:

  — If the Tag supports authenticated kill and any key has a AuthKill=1 then the Tag is killable

  — If any bits of the kill password are 1_2 then the Tag is killable

  — If **kill-pwd-read/write** (see 6.3.2.12.3.5) is 0_2 then the Tag is killable

  — If **kill-pwd-permalock** (see 6.3.2.12.3.5) is 0_2 then the Tag is killable

— **NR (Nonremovable indicator, bit $21E_h$):** If bit $21E_h$ is 0_2 then a Tag is either removable or does not support the **NR** flag. If bit $21E_h$ is 1_2 then a Tag is nonremovable. See 4.

— **H (Hazmat indicator, bit $21F_h$):** If bit $21F_h$ is 0_2 then a Tag is either not affixed to hazardous material or does not support the **H** flag. If bit $21F_h$ is 1_2 then a Tag is affixed to hazardous material.

If **T**=0 then a Tag manufacturer (if XPC_W2 exists but is not writeable) or an Interrogator (if XPC_W2 exists and is writeable) shall set all XPC_W2 bits to 0_2. If **T**=1 then a Tag and/or an Interrogator shall set the XPC_W2 bits as defined in ISO/IEC 18000-63.

The use of XPC for battery functionality is described in Clause 7.

The use of XPC for sensor functionality is described in Clauses7.6 and 8.5.1.

6.3.2.1.3 TID Memory

TID memory locations 00_h to 07_h shall contain either an $E0_h$ or $E2_h$ ISO/IEC 15963 class-identifier value. The Tag manufacturer assigns the class identifier ($E0_h$ or $E2_h$), for which ISO/IEC 15963 defines the registration authority. The class-identifier does not specify the Application. TID memory locations above 07_h shall be defined according to the registration authority defined by this class-identifier value and shall contain, at a minimum, sufficient information for an Interrogator to uniquely identify the custom commands and/or optional features that a Tag supports. TID memory may also contain Tag- and manufacturer-specific data (for example, a Tag serial number).

If the class identifier is $E0_h$ then TID memory locations 08_h to $0F_h$ contain an 8-bit manufacturer identifier, TID memory locations 10_h to $3F_h$ contain a 48-bit Tag serial number (assigned by the Tag manufacturer), the composite 64-bit TID (i.e. TID memory 00_h to $3F_h$) is unique among all classes of Tags defined in ISO/IEC 15963, and TID memory is permalocked at the time of manufacture.

If the class identifier is $E2_h$ then TID memory above 07_h shall be configured as follows:

— 08_h: XTID (**X**) indicator (whether a Tag implements an XTID – see 5.2)

— 09_h: Security (**S**) indicator (whether a Tag supports the *Authenticate* and/or *Challenge* commands)

— $0A_h$: File (**F**) indicator (whether a Tag supports the *FileOpen* command)

— $0B_h$ to 13_h: A 9-bit Tag mask-designer identifier (obtainable from the registration authority)

— 14_h to $1F_h$: A Tag-manufacturer-defined 12-bit Tag model number

— Above $1F_h$: As defined in the GS1 EPC Tag Data Standard

If the class identifier is $E2_h$ then TID memory locations 00_h to $1F_h$ shall be permalocked at time of manufacture. If the Tag implements an XTID then the entire XTID shall also be permalocked at time of manufacture.

NOTE: Some IC manufacturers use the eight MSBs of the Tag serial number for Allocation Classes E0h as a model number.

6.3.2.1.4 User Memory

A Tag may support User memory, configured as one or more files. User memory allows user data storage.

If File_0 of User memory exists and has not yet been written then the 5 LSBs of the first byte (i.e. File_0 memory addresses 03_h to 07_h) shall have the default value 00000_2.

6.3.2.1.4.1 User memory for a GS1 EPCglobal™ Application

If a Tag implements User memory then the file encoding shall be as defined in the GS1 EPC Tag Data Standard.

6.3.2.1.4.2 User memory for an ISO Application

If a Tag implements User memory then the file encoding shall be as defined in ISO/IEC 15961 and 15962.

6.3.2.2 Sessions and inventoried flags

Interrogators shall support and Tags shall provide 4 sessions (denoted S0, S1, S2, and S3). Tags shall participate in one and only one session during an inventory round. Two or more Interrogators can use sessions to independently inventory a common Tag population. The sessions concept is illustrated in Figure 6.20.

A Tag shall maintain an independent **inventoried** flag for each of its four sessions. Each **inventoried** flag has two values, denoted *A* and *B*. At the beginning of each and every inventory round an Interrogator chooses to inventory either *A* or *B* Tags in one of the four sessions. Tags participating in an inventory round in one session shall neither use nor modify an **inventoried** flag for a different session. The **inventoried** flags are the only resource that a Tag provides separately and independently to a session; all other Tag resources are shared among sessions.

After singulating a Tag an Interrogator may issue a command that causes the Tag to invert its **inventoried** flag for that session (i.e. *A*→*B* or *B*→*A*).

The following example illustrates how two Interrogators can use sessions and **inventoried** flags to independently and completely inventory a common Tag population, on a time-interleaved basis:

- Interrogator #1 powers-on, then
  - It initiates an inventory round during which it singulates *A* Tags in session S2 to *B,*
  - It powers off.
- Interrogator #2 powers-on, then
  - It initiates an inventory round during which it singulates *B* Tags in session S3 to *A,*
  - It powers off.

This process repeats until Interrogator #1 has placed all Tags in session S2 into *B*, after which it inventories the Tags in session S2 from *B* back to *A*. Similarly, Interrogator #2 places all Tags in session S3 into *A*, after which it inventories the Tags in session S3 from *A* back to *B*. By this multi-step procedure each Interrogator can independently inventory all Tags in its field, regardless of the initial state of their **inventoried** flags.

A Tag's **inventoried** flags shall have the set and persistence times shown in Table 6.20. A Tag shall power-up with its **inventoried** flags set as follows:

- The S0 **inventoried** flag shall be set to *A*.
- The S1 **inventoried** flag shall be set to either *A* or *B*, depending on its stored value, unless the flag was set longer in the past than its persistence time, in which case the Tag shall power-up with its S1 **inventoried** flag set to *A*. Because the S1 **inventoried** flag is not automatically refreshed, it may revert from *B* to *A* even when the Tag is powered.
- The S2 **inventoried** flag shall be set to either *A* or *B*, depending on its stored value, unless the Tag has lost power for a time greater than its persistence time, in which case the Tag shall power-up with the S2 **inventoried** flag set to *A*.
- The S3 **inventoried** flag shall be set to either *A* or *B*, depending on its stored value, unless the Tag has lost power for a time greater than its persistence time, in which case the Tag shall power-up with its S3 **inventoried** flag set to *A*.

A Tag shall refresh its S2 and S3 flags while powered, meaning that every time a Tag loses power its S2 and S3 **inventoried** flags shall have the set and persistence times shown in Table 6.20.

A Tag shall not change the value of its S1 **inventoried** flag from *B* to *A*, as the result of a persistence timeout, while the Tag is participating in an inventory round, is in the midst of being inventoried, or is in the midst of being accessed. If a Tag's S1 flag persistence time expires during an inventory round

then the Tag shall change the flag to *A* only (i) as instructed by an Interrogator (e.g. by a *QueryAdjust* or *QueryRep* with matching session at the end of an inventory or access operation), or (ii) at the end of the round (e.g. upon receiving a *Select* or *Query*). In case (i), if the Tag's S1 flag persistence time expires while the Tag is in the midst of being inventoried or accessed then the Tag shall change the flag to *A* at the end of the inventory or access operation. In case (ii), the Tag shall invert its S1 flag prior to evaluating the *Select* or *Query*.

6.3.2.3 Selected flag

A Tag shall implement a selected flag, **SL**, which an Interrogator may assert or deassert using a *Select* command. The Sel parameter in the *Query* command allows an Interrogator to inventory Tags that have **SL** either asserted or deasserted (i.e. **SL** or ~**SL**), or to ignore the flag and inventory Tags regardless of their **SL** value. **SL** is not associated with any particular session; **SL** may be used in any session, and is common to all sessions.

A Tag's **SL** flag shall have the set and persistence times shown in Table 6.20. A Tag shall power-up with its **SL** flag either asserted or deasserted, depending on the stored value, unless the Tag has lost power for a time greater than the **SL** persistence time, in which case the Tag shall power-up with its **SL** flag deasserted (set to ~**SL**). A Tag shall refresh its **SL** flag when powered, meaning that every time a Tag loses power its **SL** flag shall have the persistence times shown in Table 6.20.

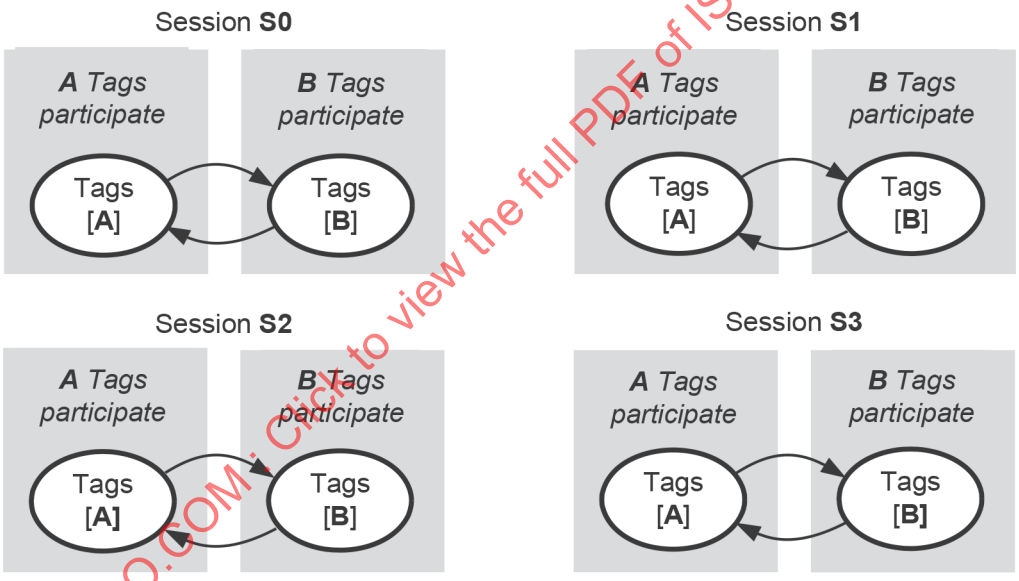


Figure 6.20 — Session diagram

6.3.2.4 C flag

A Tag's **C** flag (see 6.3.2.1.2.5) shall have the set and persistence times shown in Table 6.20. A Tag retains data in its ResponseBuffer (see 6.3.1.6.4) with the same persistence as its **C** flag. A Tag shall refresh its **C** flag when powered, meaning that every time a Tag loses power its **C** flag shall have the persistence shown in Table 6.20 (of course, if a Tag has a zero-second persistence time then even if the Tag powers down momentarily its **C** flag will be deasserted).

6.3.2.5 Security timeout

A Tag may implement a security timeout after a failed *Access*-command sequence, authenticated *Kill*, password-based *Kill*-command sequence, *Challenge*, *Authenticate*, *SecureComm*, *AuthComm*, and/or *KeyUpdate*. During a security timeout a Tag may participate in an inventory round and access, but until the end of the timeout the Tag does not execute those commands for which it implements a security timeout and instead backscatters an error code (see Annex I). If a Tag implements a security timeout

then it shall use a single timeout timer, so a security timeout caused by one command failure (such as a failed *Challenge*) shall cause a Tag to disallow all commands for which the Tag implements a security timeout until the end of the timeout period. Although this protocol gives Tag manufacturers the option of choosing which commands are subject to a security timeout, it recommends that Tags implement a security timeout at least for the *Access*-command sequence. This protocol further recommends that a Tag's security timer have the set and persistence times shown in Table 6.20.

6.3.2.6 Tag states and slot counter

A Tag shall implement the states and slot counter shown in Figure 6.21. Note that the states in Figure 6.21 are metastates that characterize a Tag's behavior and response to Interrogator commands; an actual Tag realization is likely to have more internal states than the metastates shown in the Figure 6.21. Annex B shows the associated state-transition tables; Annex C shows the associated command-response tables.

6.3.2.6.1 Ready state

Tags shall implement a **ready** state. **Ready** can be viewed as a "holding state" for energized Tags that are neither killed nor currently participating in an inventory round. Upon entering an energizing RF field a Tag that is not killed shall enter **ready**. The Tag shall remain in **ready** until it receives a *Query* command (see 6.3.2.12.2.1) whose inventoried parameter (for the session specified in the *Query*) and sel parameter match its current flag values. Matching Tags shall draw a *Q*-bit number from their RNG (see 6.3.2.7), load this number into their slot counter, and transition to the **arbitrate** state if the number is nonzero, or to the **reply** state if the number is zero. If a Tag in any state except **killed** loses power then it shall return to **ready** upon regaining power.

Table 6.20 — Tag flags and persistence values

Flag	Time to Set	Required persistence
S0 **inventoried** flag	≤ 2ms regardless of initial or final value [3]	Tag energized: Indefinite Tag not energized: None
S1 **inventoried** flag [1]	≤ 2ms regardless of initial or final value [3]	Tag energized: Nominal temperature range: 500ms < persistence < 5s Extended temperature range: Not specified Tag not energized: Nominal temperature range: 500ms < persistence < 5s Extended temperature range: Not specified
S2 **inventoried** flag [1]	≤ 2ms regardless of initial or final value [3]	Tag energized: Indefinite Tag not energized: Nominal temperature range: 2s < persistence Extended temperature range: Not specified
S3 **inventoried** flag [1]	≤ 2ms regardless of initial or final value [3]	Tag energized: Indefinite Tag not energized: Nominal temperature range: 2s < persistence Extended temperature range: Not specified
Selected (**SL**) flag [1]	≤ 2ms regardless of initial or final value [3]	Tag energized: Indefinite Tag not energized: Nominal temperature range: 2s < persistence Extended temperature range: Not specified

Table 6.20 *(continued)*

Flag	Time to Set	Required persistence
C flag [1,2]	Deassert: ≤ 2ms [3] Assert: ≤ 0ms measured relative to the first rising edge of the Tag's response indicating that the Tag has finished its computation	Tag energized: Indefinite Tag not energized: Nominal temperature range: 0s ≤ persistence < 5s Extended temperature range: Not specified
Optional security timeout	< $T_{1(min)}$ (see Table 6.16), measured relative to the last rising edge of the last bit of the Interrogator command that caused the security timeout.	Tag energized [4]: Nominal temperature range: 20ms ≤ persistence < 200ms Extended temperature range: Not specified Tag not energized [4]: Nominal temperature range: 20ms ≤ persistence < 200ms Extended temperature range: Not specified

Note 1 For a randomly chosen and sufficiently large Tag population, 95% of the Tag persistence times shall meet the persistence requirement, with a 90% confidence interval.

Note 2 A Tag retains data in its ResponseBuffer with the same persistence as its C flag (see 6.3.1.6.4).

Note 3 Measured from the last rising edge of the last bit of the Interrogator transmission that caused the change.

Note 4 The indicated persistence times are recommended but not required.

6.3.2.6.2 Arbitrate state

Tags shall implement an **arbitrate** state. **Arbitrate** can be viewed as a "holding state" for Tags that are participating in the current inventory round but whose slot counters (see 6.3.2.12.2.3) hold nonzero values. A Tag in **arbitrate** shall decrement its slot counter every time it receives a *QueryRep* command (see 6.3.2.6.8) whose session parameter matches the session for the inventory round currently in progress, and it shall transition to the **reply** state and backscatter an RN16 when its slot counter reaches 0000_h. Tags that return to **arbitrate** (for example, from the **reply** state) with a slot value of 0000_h shall decrement their slot counter from 0000_h to $7FFF_h$ at the next *QueryRep* (with matching session) and, because their slot value is now nonzero, shall remain in **arbitrate**.

6.3.2.6.3 Reply state

Tags shall implement a **reply** state. Upon entering **reply** a Tag shall backscatter an RN16. If the Tag receives a valid acknowledgement (*ACK*) then it shall transition to the **acknowledged** state, backscattering the reply shown in Table 6.17. If the Tag fails to receive an *ACK* within time $T_{2(max)}$, or receives an invalid *ACK* or an *ACK* with an erroneous RN16 then it shall return to **arbitrate**. Tag and Interrogator shall meet all timing requirements specified in Table 6.16.

6.3.2.6.4 Acknowledged state

Tags shall implement an **acknowledged** state. A Tag in **acknowledged** may transition to any state except **killed**, depending on the received command (see Figure 6.21). If a Tag in the **acknowledged** state receives a valid *ACK* containing the correct RN16 then it shall re-backscatter the reply shown in Table 6.17. If a Tag in the **acknowledged** state fails to receive a valid command within time $T_{2(max)}$ then it shall return to **arbitrate**. Tag and Interrogator shall meet all timing requirements specified in Table 6.16.

6.3.2.6.5 Open state

Tags shall implement an **open** state. A Tag in the **acknowledged** state whose access password is nonzero shall transition to **open** upon receiving a *Req_RN* command, backscattering a new RN16 (denoted handle) that the Interrogator shall use in subsequent commands and the Tag shall use in subsequent replies. A Tag in the **open** state may execute some access commands – see

Table 6.27. A Tag in **open** may transition to any state except **acknowledged**, depending on the received command (see Figure 6.21). If a Tag in the **open** state receives a valid *ACK* containing the correct handle then it shall re-backscatter the reply shown in Table 6.17. Tag and Interrogator shall meet all timing requirements specified in Table 6.16 except $T_{2(max)}$; in the **open** state the maximum delay between Tag response and Interrogator transmission is unrestricted.

6.3.2.6.6 Secured state

Tags shall implement a **secured** state. A Tag in the **acknowledged** state whose access password is zero shall transition to **secured** upon receiving a *Req_RN* command, backscattering a new RN16 (denoted handle) that the Interrogator shall use in subsequent commands and the Tag shall use in subsequent replies. A Tag in the **open** state shall transition to **secured** following a successful *Access* command sequence or Interrogator authentication (where success in the latter case is defined by the cryptographic suite specified in the *Authenticate* command that initiated the authentication), maintaining the same handle that it previously backscattered when it transitioned from the **acknowledged** state to the **open** state. A Tag in the **secured** state with the appropriate Tag and file privileges (see 6.3.2.11.2 and 6.3.2.11.3) may execute all access commands. A Tag in **secured** may transition to any state except **acknowledged**, depending on the received command (see Figure 6.21). If a Tag in the **secured** state receives a valid *ACK* containing the correct handle then it shall re-backscatter the reply shown in Table 6.17. Tag and Interrogator shall meet all timing requirements specified in Table 6.16 except $T_{2(max)}$; in the **secured** state the maximum delay between Tag response and Interrogator transmission is unrestricted.

6.3.2.6.7 Killed state

Tags shall implement a **killed** state. A Tag in either the **open** or **secured** state shall enter the **killed** state upon receiving a successful password-based *Kill*-command sequence with a correct nonzero kill password and handle. A Tag in the **secured** states shall enter the **killed** state upon a successful authenticated *Kill* (see 6.3.2.12.3.4). *Kill* permanently disables a Tag. Upon entering the **killed** state a Tag shall notify the Interrogator that the kill was successful and shall not respond to an Interrogator thereafter. Killed Tags shall remain in the **killed** state under all circumstances, and shall immediately enter killed upon subsequent power-ups. Killing a Tag is irreversible.

6.3.2.6.8 Slot counter

Tags shall implement a 15-bit slot counter. Upon receiving a *Query* or *QueryAdjust* command a Tag shall preload into its slot counter a value between 0 and 2^Q–1, drawn from the Tag's RNG (see 6.3.2.7). *Q* is an integer in the range (0, 15). A *Query* specifies *Q*; a *QueryAdjust* may modify *Q* from the prior *Query*.

Tags in the **arbitrate** state decrement their slot counter every time they receive a *QueryRep* with matching session, transitioning to the **reply** state and backscattering an RN16 when their slot counter reaches 0000_h. Tags whose slot counter reached 0000_h, who replied, and who were not acknowledged (including Tags that responded to an original *Query* and were not acknowledged) shall return to **arbitrate** with a slot value of 0000_h and shall decrement this slot value from 0000_h to $7FFF_h$ at the next *QueryRep*. The slot counter shall be capable of continuous counting, meaning that, after the slot counter rolls over to $7FFF_h$ it begins counting down again, thereby effectively preventing subsequent replies until the Tag loads a new random value into its slot counter. See also Annex J.

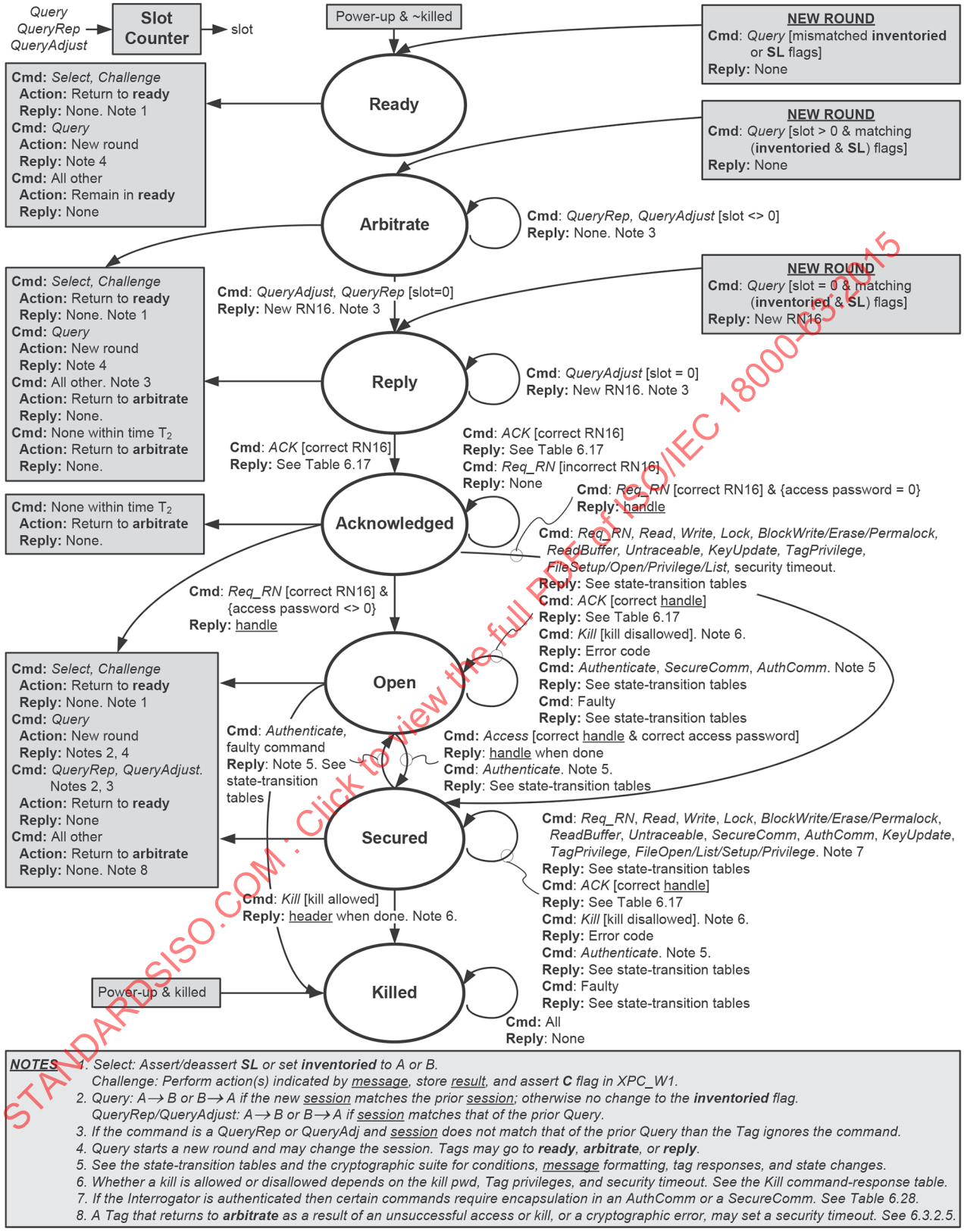


Figure 6.21 — Tag state diagram

6.3.2.7 Tag random or pseudo-random number generator

A Tag shall implement a random or pseudo-random number generator (RNG). The RNG shall meet the following randomness criteria independent of the strength of the energizing RF field, the R=>T link rate, and the data stored in the Tag (including but not limited to the StoredPC, XPC word or words, UII, and StoredCRC). Tags shall generate 16-bit random or pseudo-random numbers (RN16) using the RNG, and shall have the ability to extract *Q*-bit subsets from its RNG to preload the Tag's slot counter (see 6.3.2.6.8). Tags shall have the ability to temporarily store at least two RN16s while powered, to use, for example, as a handle and a 16-bit cover-code during password transactions (see Figure 6.24 or Figure 6.26).

Probability of a single RN16: The probability that any RN16 drawn from the RNG has value RN16 = *j*, for any *j*, shall be bounded by $0.8/2^{16} < P(RN16 = j) < 1.25/2^{16}$.

Probability of simultaneously identical sequences: For a Tag population of up to 10,000 Tags, the probability that any two or more Tags simultaneously generate the same sequence of RN16s shall be less than 0.1%, regardless of when the Tags are energized.

Probability of predicting an RN16: An RN16 drawn from a Tag's RNG 10ms after the end of T_r in Figure 6.3 shall not be predictable with a probability greater than 0.025% if the outcomes of prior draws from the RNG, performed under identical conditions, are known.

This protocol recommends that Interrogators wait 10ms after T_r in Figure 6.3 or T_{hr} in Figure 6.5 before issuing passwords to Tags.

A cryptographic suite defines RNG requirements and randomness criteria for cryptographic operations. These requirements and criteria may be different, and in particular may be more stringent, than those defined above for inventory and password operations.

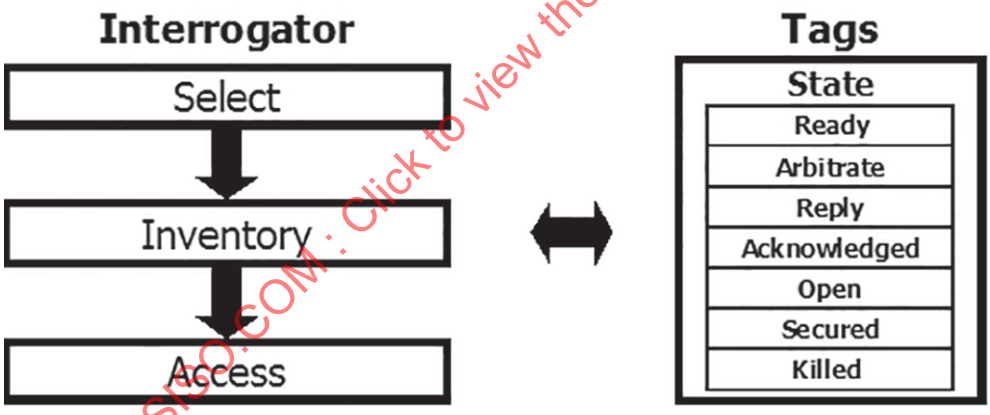


Figure 6.22 — Interrogator/Tag operations and Tag state

6.3.2.8 Managing Tag populations

Interrogators manage Tag populations using the three basic operations shown in Figure 6.22. Each of these operations comprises multiple commands. The operations are defined as follows:

a) **Select:** The process by which an Interrogator selects a Tag population for subsequent inventory or cryptographically challenges a Tag population for subsequent authentication. Select comprises the *Select* and *Challenge* commands.

b) **Inventory:** The process by which an Interrogator identifies Tags. An Interrogator begins an inventory round by transmitting a *Query* command in one of four sessions. One or more Tags may reply. The Interrogator detects a single Tag reply and requests the PC, optional XPC word(s), UII, and CRC-16 from the Tag. An inventory round operates in one and only one session at a time. Annex E shows an example of an Interrogator inventorying and accessing a single Tag. Inventory comprises multiple commands.

c) **Access:** The process by which an Interrogator transacts with (reads, writes, authenticates, or otherwise engages with) an individual Tag. An Interrogator singulates and uniquely identifies a Tag prior to access. Access comprises multiple commands.

6.3.2.9 Selecting Tag populations

The select process comprises two commands, *Select* and *Challenge*. *Select* allows an Interrogator to select a Tag population for subsequent inventorying. *Challenge* allows an Interrogator to challenge a Tag population for subsequent authentication. *Select* and *Challenge* are the only two commands that an Interrogator may issue prior to inventory, and they are not mutually exclusive (i.e. an Interrogator may issue both a *Select* and a *Challenge* prior to starting an inventory round). *Select* is a mandatory command; *Challenge* is optional.

A *Select* command allows an Interrogator to select a particular Tag population prior to inventorying. The selection is based on user-defined criteria, enabling union (U), intersection (∩), and negation (~) based Tag partitioning. Interrogators perform U and ∩ operations by issuing successive *Select* commands. *Select* can assert or deassert a Tag's **SL** flag, or it can set a Tag's **inventoried** flag to either *A* or *B* in any one of the four sessions.

Upon receiving a *Select* a not-killed Tag returns to the **ready** state, evaluates the criteria, and depending on the evaluation may modify the indicated **SL** or **inventoried** flag. A *Query* command uses these flags to choose which Tags participate in a subsequent inventory round. An Interrogator may inventory and access **SL** or ~**SL** Tags, or it may choose to not use the **SL** flag at all. *Select* may begin with a Tag in any state except **killed**, and ends with a Tag in **ready**.

Select contains the parameters Target, Action, MemBank, Pointer, Length, Mask, and Truncate.

— Target and Action indicate whether and how a *Select* modifies a Tag's **SL** or **inventoried** flag, and in the case of an **inventoried** flag, for which session. A *Select* that modifies the **SL** flag does not modify an **inventoried** flag, and vice versa.

— MemBank specifies if the Mask applies to UII, TID, or User memory. *Select* commands apply to a single memory bank. Successive *Selects* may apply to different memory banks.

— Pointer, Length, and Mask: Pointer and Length describe a memory range. Mask, which is Length bits long, contains a bit string that a Tag compares against the specified memory range.

— Truncate specifies whether a Tag backscatters its entire UII, or only that portion of the UII immediately following Mask, when replying to an *ACK*.

A *Challenge* command allows an Interrogator to instruct multiple Tags to simultaneously yet independently precompute and store a cryptographic result for use in a subsequent authentication. Because cryptographic algorithms often require significant computation time, parallel precomputation may significantly accelerate the authentication of a Tag population. *Challenge* contains an immed parameter which, if asserted, instructs the Tags to concatenate their response (result or error code) to the UII backscattered in reply to an *ACK*.

Upon receiving a *Challenge* a not-killed Tag that supports the command returns to the **ready** state, evaluates the command (including whether it supports the CSI specified in the *Challenge*), and depending on the evaluation may compute and store a cryptographic result in its ResponseBuffer. In some instances a Tag may use the stored result during a subsequent authentication. In such instances the Interrogator will transmit a subsequent *Authenticate* command (see 6.3.2.12.3.10) to the previously challenged Tag. In other instances the Tag's stored result may be usable without a subsequent *Authenticate*. For an example of the latter case, in some cryptographic suites an Interrogator can verify a Tag's authenticity simply by evaluating the precomputed result. *Challenge* may begin with a Tag in any state except **killed**, and ends with a Tag in **ready**.

6.3.2.10 Inventorying Tag populations

The inventory command set includes *Query*, *QueryAdjust*, *QueryRep*, *ACK*, and *NAK*. *Query* initiates an inventory round and decides which Tags participate in the round ("inventory round" is defined in 4).

Query contains a slot-count parameter *Q*. Upon receiving a *Query* participating Tags pick a random value in the range (0, 2^Q–1), inclusive, and load this value into their slot counter. Tags that pick a zero transition to the **reply** state and reply immediately. Tags that pick a nonzero value transition to the **arbitrate** state and await a *QueryAdjust* or *QueryRep* command. Assuming a single Tag replies, the query-response algorithm proceeds as follows:

a) The Tag backscatters an RN16 as it enters **reply**,

b) The Interrogator acknowledges the Tag with an *ACK* containing this same RN16,

c) The acknowledged Tag transitions to the **acknowledged** state, backscattering a reply as in Table 6.17,

d) The Interrogator issues a *QueryAdjust* or *QueryRep,* causing the identified Tag to invert its **inventoried** flag (i.e. *A*→*B* or *B*→*A*) and transition to **ready**, and potentially causing another Tag to initiate a query-response dialog with the Interrogator, starting in step (a), above.

If the Tag fails to receive the *ACK* in step (b) within time T_2 (see Figure 6.18), or receives the *ACK* with an erroneous RN16, then it returns to **arbitrate**.

If multiple Tags reply in step (a) but the Interrogator, by detecting and resolving collisions at the waveform level, can resolve an RN16 from one of the Tags, the Interrogator can *ACK* the resolved Tag. Unresolved Tags receive erroneous RN16s and return to **arbitrate** without backscattering the reply shown in Table 6.17.

If the Interrogator sends a valid *ACK* (i.e. an *ACK* containing the correct RN16) to the Tag in the **acknowledged** state, the Tag re-backscatters the reply shown in Table 6.17.

At any point the Interrogator may issue a *NAK*, in response to which all Tags in the inventory round that receive the *NAK* return to **arbitrate** without changing their **inventoried** flag.

After issuing a *Query* to initiate an inventory round, the Interrogator typically issues one or more *QueryAdjust* or *QueryRep* commands. *QueryAdjust* repeats a previous *Query* and may increment or decrement *Q*, but does not introduce new Tags into the round. *QueryRep* repeats a previous *Query* without changing any parameters and without introducing new Tags into the round. An inventory round can contain multiple *QueryAdjust* or *QueryRep* commands. At some point the Interrogator will issue a new *Query*, thereby starting a new inventory round.

Tags in the **arbitrate** or **reply** states that receive a *QueryAdjust* first adjust *Q* (increment, decrement, or leave unchanged), then pick a random value in the range (0, 2^Q–1), inclusive, and load this value into their slot counter. Tags that pick zero transition to the **reply** state and reply immediately. Tags that pick a nonzero value transition to the **arbitrate** state and await a *QueryAdjust* or a *QueryRep* command.

Tags in the **arbitrate** state decrement their slot counter every time they receive a *QueryRep*, transitioning to the **reply** state and backscattering an RN16 when their slot counter reaches 0000_h. Tags whose slot counter reached 0000_h, who replied, and who were not acknowledged (including Tags that responded to the original *Query* and were not acknowledged) return to **arbitrate** with a slot value of 0000_h and decrement this slot value from 0000_h to $7FFF_h$ at the next *QueryRep*, thereby effectively preventing subsequent replies until the Tag loads a new random value into its slot counter.

Although Tag inventory is based on a random protocol, the *Q*-parameter affords network control by allowing an Interrogator to regulate the probability of Tag responses. *Q* is an integer in the range (0,15); thus, the associated Tag-response probabilities range from $2^0 = 1$ to $2^{-15} = 0.000031$.

Annex D describes an exemplary Interrogator algorithm for choosing *Q*.

The scenario outlined above assumed a single Interrogator operating in a single session. However, as described in 6.3.2.2, an Interrogator can inventory a Tag population in one of four sessions. Furthermore,

as described in 6.3.2.12.2, the *Query*, *QueryAdjust*, and *QueryRep* commands each contain a session parameter. How a Tag responds to these commands varies with the command, session parameter, and Tag state, as follows:

— *Query*: A *Query* command starts an inventory round and chooses the session for the round. Tags in any state except **killed** execute a *Query*, starting a new round in the specified session and transitioning to **ready**, **arbitrate**, or **reply**, as appropriate (see Figure 6.21).

  — If a Tag in the **acknowledged**, **open**, or **secured** states receives a *Query* whose session parameter matches the prior session then it inverts its **inventoried** flag (i.e. *A*→*B* or *B*→*A*) for the session before it evaluates whether to transition to **ready**, **arbitrate**, or **reply**.

  — If a Tag in the **acknowledged**, **open**, or **secured** states receives a *Query* whose session parameter does not match the prior session then it leaves its **inventoried** flag for the prior session unchanged as it evaluates whether to transition to **ready**, **arbitrate**, or **reply**.

— *QueryAdjust, QueryRep*: Tags in any state except **ready** or **killed** execute a *QueryAdjust* or *QueryRep* command if, and only if, (i) the session parameter in the command matches the session parameter in the *Query* that started the round, and (ii) the Tag is not in the middle of a *Kill* or *Access* command sequence (see 6.3.2.12.3.4 or 6.3.2.12.3.6, respectively). Tags ignore a *QueryAdjust* or *QueryRep* with mismatched session.

  — If a Tag in the **acknowledged**, **open**, or **secured** states receives a *QueryAdjust* or *QueryRep* whose session parameter matches the session parameter in the prior *Query*, and the Tag is not in the middle of a *Kill* or *Access* command sequence (see 6.3.2.12.3.4 or 6.3.2.12.3.6, respectively), then it inverts its **inventoried** flag (i.e. *A*→*B* or *B*→*A*) for the current session and then transitions to **ready**.

To illustrate an inventory operation, consider a specific example: Assume a population of 64 powered Tags in the **ready** state. An Interrogator first issues a *Select* to select a subpopulation of Tags. Assume that 16 Tags match the selection criteria. Further assume that 12 of the 16 selected Tags have their **inventoried** flag set to *A* in session S0. The Interrogator issues a *Query* specifying (**SL**, *Q* = 4, S0**,** *A*). Each of the 12 Tags picks a random number in the range (0,15) and loads the value into its slot counter. Tags that pick a zero respond immediately. The *Query* has 3 possible outcomes:

a) **No Tags reply:** The Interrogator may issue another *Query*, or it may issue a *QueryAdjust* or *QueryRep*.

b) **One Tag replies** (see Figure 6.23): The Tag transitions to the **reply** state and backscatters an RN16. The Interrogator acknowledges the Tag by sending an *ACK*. If the Tag receives the *ACK* with a correct RN16 it backscatters the reply shown in Table 6.17 and transitions to the **acknowledged** state. If the Tag receives the *ACK* with an incorrect RN16 it transitions to **arbitrate**. Assuming a successful *ACK*, the Interrogator may either access the acknowledged Tag or issue a *QueryAdjust* or *QueryRep* with matching session parameter to invert the Tag's **inventoried** flag from *A*→*B* and send the Tag to **ready** (a *Query* with matching prior-round session parameter will also invert the **inventoried** flag from *A*→*B*).

c) **Multiple Tags reply:** The Interrogator observes a backscattered waveform comprising multiple RN16s. It may try to resolve the collision and issue an *ACK*; not resolve the collision and issue a *QueryAdjust, QueryRep*, or *NAK*; or quickly identify the collision and issue a *QueryAdjust* or *QueryRep* before the collided Tags have finished backscattering. In the latter case the collided Tags, not observing a valid reply within T_2 (see Figure 6.18), return to **arbitrate** and await the next *Query* or *QueryAdjust* command.

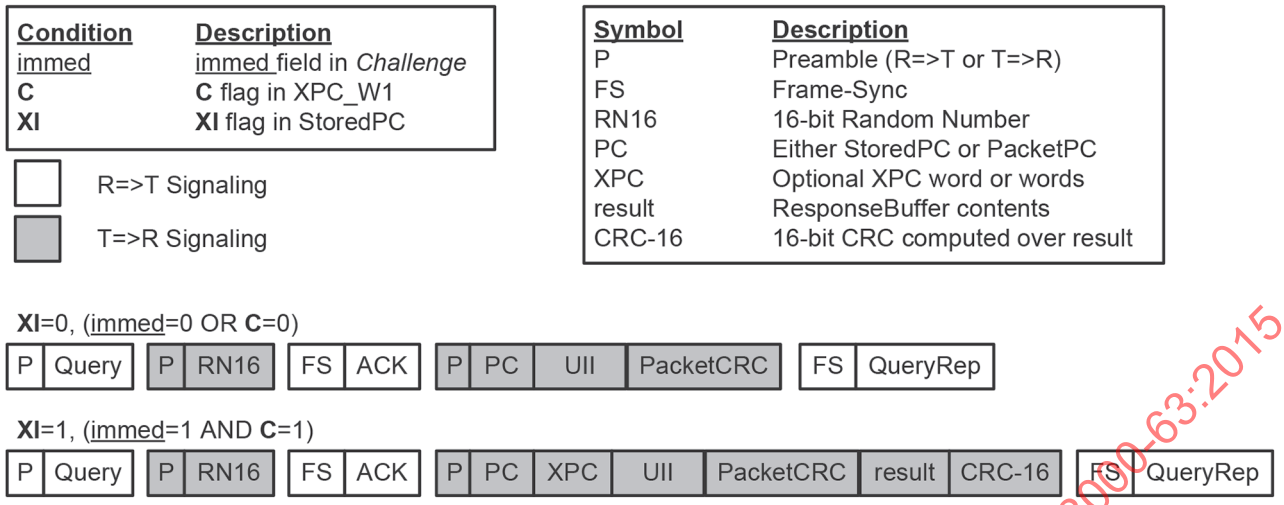


Figure 6.23 — One Tag reply

6.3.2.11 Accessing individual Tags

An Interrogator may choose to access a Tag after acknowledging it. The access commands are *Req_RN*, *Read*, *Write*, *Lock*, *Kill*, *Access*, *BlockWrite*, *BlockErase*, *BlockPermalock*, *Authenticate*, *ReadBuffer*, *SecureComm*, *AuthComm*, *KeyUpdate*, *Untraceable*, *FileOpen*, *FileList*, *FilePrivilege*, *FileSetup*, and *TagPrivilege*. A Tag shall execute access commands only in the states shown in

Table 6.27. A Tag shall treat as invalid (see Table C.30) optional access commands that it does not support. See Annex K for an example of a data-flow exchange during which an Interrogator accesses a Tag and reads its kill password.

Access always begins with an Interrogator moving a Tag from the **acknowledged** state to either the **open** or the **secured** state as follows:

Step 1: The Interrogator issues a *Req_RN* to the acknowledged Tag.

Step 2: The Tag generates and stores a new RN16 (denoted handle), backscatters the handle, and transitions to the **open** state if its access password is nonzero, or to the **secured** state if its access password is zero. The Interrogator may now issue further access commands.

All access commands include a Tag's handle. Upon receiving an access command a Tag verifies that the handle is correct prior to executing the command, and does not execute access commands with an incorrect handle. The handle value is fixed for the entire duration of a Tag access.

An Interrogator may issue an *ACK* to a Tag in the **open** or **secured** states, with the Tag's handle as the RN in the command, thereby causing the Tag to backscatter the reply shown in Table 6.17. A Tag in the **open** or **secured** states that receives an *ACK* with an incorrect handle transitions to the **arbitrate** state without replying and without changing its inventoried flag.

As shown in Table 6.27, some access commands require a prior *Req_RN* and some a prior authentication before execution. A Tag's response to an access command includes, at a minimum, the Tag's handle; the response may include other information as well (for example, the result of a *Read*). An Interrogator shall verify the correctness of the handle in a Tag's response to an access command.

The *Authenticate* and *Access* commands provide the only means to transition a Tag from the **open** state to the **secured** state. The *Authenticate* command or a faulty security command provide the only means to transition a Tag from the **secured** state back to the **open** state. See Table C.18 and Table C.30.

The privileges that a Tag in the **open** state grants to an Interrogator depend on the authorization level of the **open** state. The privileges that a Tag in the **secured** state grants to an Interrogator depend on the

authorization level of the access or authentication that most recently moved the Tag to that state. An Interrogator that moved a Tag to the **secured** state using one means (for example, an *Access* command) may later cause the Tag to re-enter the **secured** state using a different means (for example, an *Authenticate* command), affording the Interrogator different privileges. See 6.3.2.11.2 for a discussion of privileges and keys.

A Tag may enter the **secured** state by means of a:

— *Req_RN*: If a Tag's access password is zero then the Tag transitions from the **acknowledged** state to the **secured** state at the beginning of access (i.e. upon receiving a *Req_RN*), bypassing the **open** state.

— *Access*: A Tag whose access password is nonzero transitions from the **open** or **secured** state to the **secured** state upon successfully executing an *Access*-command sequence.

— *Authenticate*: A Tag transitions from the **open** or **secured** state to the **secured** state upon successfully executing an Interrogator or mutual authentication.

A Tag may limit an Interrogator's access to the **secured** state via one or more physical mechanisms. For example, a Tag may require that its received RF power exceed a threshold before it will enter the **secured** state. This protocol does not specify such physical mechanisms but allows them at a Tag manufacturer's discretion.

An Interrogator and a Tag can communicate indefinitely in the **open** or **secured** states. The Interrogator may end the communications at any time by issuing a *Select*, *Challenge*, *Query*, *QueryAdjust*, *QueryRep*, or *NAK*. The Tag's response to a *Query*, *QueryAdjust*, or *QueryRep* is described in 6.3.2.8. A *NAK* causes all Tags in the inventory round to return to **arbitrate** without changing their **inventoried** flag(s).

Interrogators in some regulatory regions are required to hop frequency at periodic intervals, ending the inventory round and any access operations. Unfortunately, some cryptographic operations take longer than a hop interval to complete. This protocol allows a cryptographic suite to specify that a Tag retain one or more cryptographic state variables during a temporary power loss such as a frequency hop, and allows an Interrogator to re-acquire the Tag in a subsequent inventory round and resume the cryptographic operation.

This protocol recommends that Interrogators avoid powering-off while a Tag is in the **reply**, **acknowledged**, **open** or **secured** states. Rather, Interrogators should end (or in the case of a long cryptographic operation, suspend) their dialog with a Tag before powering off, leaving the Tag in either the **ready** or **arbitrate** state.

This protocol partitions the access commands into the subclasses Core, Security, and File Management (see also

Table 6.27). The purpose of this subclass partitioning is solely for ease of discussion and the particular subclass does not convey or deny requirements to or from any access command.

6.3.2.11.1 Core access commands

The core access commands are *Req_RN*, *Read*, *Write*, *Lock*, *Kill*, *Access*, *BlockWrite*, *BlockErase*, *BlockPermalock*, and *Untraceable*. *Req_RN*, *Read*, *Write*, *Lock*, and *Kill* are mandatory. *Access*, *BlockWrite*, *BlockErase*, *BlockPermalock*, and *Untraceable* are optional. A Tag may implement one or more of the optional commands regardless of whether the Tag supports cryptographic security or file management.

A *Req_RN* command allows an Interrogator to (a) transition a Tag from the **acknowledged** state to the **open** or **secured** states, obtaining the Tag's handle in the process, or (b) ask a Tag in the **open** or **secured** states to backscatter a 16-bit random number.

A *Read* command allows an Interrogator to read Tag memory. An Interrogator may read a Tag's kill and/or access passwords depending on Tag state and the password's lock status. An Interrogator with an asserted Untraceable privilege may read UII and TID memory, and User-memory files for which it has read privileges. An Interrogator with a deasserted Untraceable privilege may read the portions of

UII and TID memory that are not untraceably hidden and may read User-memory files for which it has read privileges if User memory is not untraceably hidden.

The *Write*, *BlockWrite*, and *BlockErase* commands allow an Interrogator to write or erase portions of Tag memory. Whether, in what states, and with what privileges an Interrogator may write/erase Tag memory is described in Table 6.24, Table 6.25, and Table 6.27.

The *Lock* and *BlockPermalock* commands allow an Interrogator to configure portions of Tag memory to be changeably or permanently writeable or unwriteable. The UII memory bank, TID memory bank, File_0, and the access and kill passwords may be unlocked, permanently unlocked, locked, or permanently locked for writing. Blocks within File_0 and File_N (N>0) may also be unlocked or permanently locked for writing. An *Untraceable* is the only command that can write to permanently locked memory, but its writing ability is limited to the **L** and **U** bits in UII memory (see 6.3.2.12.3.16).

An *Access* command allows an Interrogator to transition a Tag from the **open** to the **secured** state. The transition is a multi-step procedure described in 6.3.2.12.3.6 and outlined in Figure 6.26, in which an Interrogator sends two successive *Access* commands to a Tag. The first *Access* command contains the first half of the access password; the second *Access* command contains the second half. If a Tag receives a properly formatted *Access*-command sequence with the correct access password then it transitions to the **secured** state.

Table 6.21 — Conditions for Killing a Tag

Tag supports authenticated kill?	Kill Pwd	Kill-Pwd Locked?	Tag Killable?	How?
No	Zero	Permalocked	No	–
		Locked, unlocked, per-maunlocked	Yes	Write nonzero kill pwd then use *Kill* command. If kill pwd is locked and access pwd is nonzero then requires access before writing new kill pwd
	Nonzero	All	Yes	Use *Kill* command with kill pwd.
Yes (and at least one key has Auth-Kill=1)	Zero	Permalocked	Yes	Authenticate Interrogator then perform authenticated kill.
		Locked, unlocked, per-maunlocked	Yes	Authenticate Interrogator then perform authenticated kill, or write nonzero kill pwd then use *Kill* command. If kill pwd is locked and access pwd is nonzero then requires access before writing new kill pwd
	Nonzero	All	Yes	Authenticate Interrogator then perform authenticated kill or use *Kill* command with kill pwd.

The *Untraceable* command allows an Interrogator with an asserted Untraceable privilege (Table 6.22 and Table 6.23) to instruct a Tag to (a) overwrite the **L** bits in its StoredPC and **U** bit in XPC_W1, (b) hide part of its memory from Interrogators with a deasserted Untraceable privilege and/or (c) reduce its operating range. An Interrogator may use the **U** bit of XPC_W1, if supported, to indicate whether a Tag is hiding memory and/or is reducing its operating range (collectively, untraceable). An untraceable Tag behaves identically, from a command-response and state-machine perspective, to a traceable Tag, but behaves as though portions of its memory do not exist and/or as though it has reduced sensitivity. An untraceable Tag does not erase hidden memory; an Interrogator with an asserted Untraceable privilege may subsequently reexpose untraceably hidden memory to all Interrogators and/or reenable full operating range. An Interrogator may also subsequently overwrite the **L** and **U** bits.

A *Kill* command allows an Interrogator to kill a Tag. If a Tag's kill password is nonzero then an Interrogator may kill the Tag using the multi-step password-based *Kill*-command sequence shown in Figure 6.24. If a Tag supports authenticated killing then an Interrogator that authenticated itself using a key with an AuthKill privilege (see Table 6.23) may kill the Tag regardless of its kill-password value (zero or nonzero) using the abbreviated, authenticated kill process shown in Figure 6.24. A Tag

that does not implement a kill password, or whose kill password is zero, is not killable except by the authenticated kill process. A successful *Kill* moves a Tag from the **open** or **secured** state to the **killed** state. A Tag, once killed, shall not respond to an Interrogator thereafter.

To minimize the risk of illicit Tag killing, this protocol recommends that killable Tags use either (1) unique kill passwords or (2) permalocked zero-valued kill passwords and authenticated kill. This protocol also recommends against a zero-valued access password.

The **K** flag of XPC_W1 indicates whether a Tag is killable. As shown in Table 6.21, the only situation in which a Tag is not killable by over-the-air commands is if the Tag has a permalocked, zero-valued kill password and either does not support authenticated kill or does not grant the AuthKill privilege to any key.

The *Write*, *Kill*, and *Access* commands send 16-bit words (either data or half-passwords) from Interrogator to Tag using one-time-pad-based link cover coding to obscure the word being transmitted, as follows:

Step 1: The Interrogator issues a *Req_RN,* to which the Tag responds by backscattering a new RN16. The Interrogator then generates a 16-bit string comprising a bit-wise EXOR of the 16-bit word to be transmitted with this new RN16, both MSB first, and issues the command with this string as a parameter.

Step 2: The Tag recovers the 16-bit word by performing a bit-wise EXOR of the received 16-bit string with the original RN16.

If an Interrogator issues a command containing cover-coded data or half-password and fails to receive a response from the Tag then the Interrogator may subsequently reissue the command unchanged. If the Interrogator issues a subsequent command containing new data or a new half-password then it shall first issue a *Req_RN* to obtain a new RN16 and shall use this new RN16 for the cover-coding.

The *BlockWrite* command (see 6.3.2.12.3.7) communicates multiple 16-bit words from Interrogator to Tag. Unlike a *Write*, *BlockWrite* does not use link cover coding.

Although the *Access* command uses a password, an *Access*-command sequence is not cryptographically secure. Neither Tag nor Interrogator shall consider themselves authenticated following an *Access*-command sequence. A Tag or an Interrogator shall only consider themselves authenticated after executing a cryptographic authentication in accordance with a cryptographic suite.

6.3.2.11.2 Security access commands

The security access commands are *Authenticate*, *SecureComm*, *AuthComm*, *KeyUpdate*, and *TagPrivilege*. All are optional. A Tag may implement one or more of these commands regardless of whether the Tag supports optional core commands and/or file management. Some of these commands require prior authentication.

An *Authenticate* command may implement Tag, Interrogator, and/or mutual authentication, depending on the Tag's implementation of the cryptographic suite specified by CSI in the command. Authentication may include deriving session keys and exchanging parameters for subsequent communications. Depending on the cryptographic suite, the message field in the *Authenticate* command may include a KeyID, the type of authentication, and for some multi-step authentications the step number in the authentication sequence.

An *AuthComm* command allows authenticated R=>T communications. Table 6.28 shows which commands an Interrogator may, and an authenticated Interrogator shall, encapsulate in an *AuthComm*. An *AuthComm* protects communications according to the cryptographic suite specified by CSI in the *Challenge* or *Authenticate* that preceded the *AuthComm*.

A *SecureComm* command allows secure R=>T communications. Table 6.28 shows which commands an Interrogator may, and an authenticated Interrogator shall, encapsulate in a *SecureComm*. A *SecureComm* protects communications according to the cryptographic suite specified by CSI in the *Challenge* or *Authenticate* that preceded the *SecureComm*.

A *SecureComm* is configured to allow more robust security than an *AuthComm*; an *AuthComm* is configured to be faster with simplified Tag processing. This protocol recommends that Interrogators not intermix *SecureComm* and *AuthComm* commands when engaging in an authenticated dialog with a Tag.

A *KeyUpdate* command allows an authenticated Interrogator to write or change a key. If a Tag does not write the new key successfully then it defaults to the prior stored key. An Interrogator may use a *KeyUpdate* to change the key that it used during authentication; if the Interrogator has an asserted CryptoSuperuser privilege (see Table 6.23) then it may also change value(s) for other key(s) in the cryptographic suite. A cryptographic suite may place additional restrictions, beyond those specified in this protocol, on when and whether a key may be updated.

A *TagPrivilege* command allows an Interrogator to read or modify the privileges in Table 6.22 or Table 6.23 for the access password or for a key, respectively. Whether a Tag executes a *TagPrivilege* depends on the privilege level of the access password or the key that the Interrogator supplied during the access or authentication.

A Tag may support zero, one, or more than one cryptographic suite(s). A cryptographic suite defines how a Tag and an Interrogator implement a cryptographic algorithm and its functions. The Tag manufacturer shall choose the number and type of cryptographic suites that a Tag supports; this assignment shall not be alterable in the field. An Interrogator selects one from among the implemented cryptographic suites using the CSI field in the *Challenge* and *Authenticate* commands.

A Tag may support up to 256 keys, numbered Key_0 to Key_255. The Tag manufacturer shall choose the number of available keys and assign them to the cryptographic suite(s); this assignment shall not be alterable in the field. No two keys shall have the same number, even if used for different cryptographic suites. A Tag shall not indicate where in memory it stores its keys, nor shall it allow an Interrogator to read this memory location.

Although the functions and security of cryptographic suites may vary, this protocol anticipates that some suites may perform only Tag authentication, whereas others may perform Interrogator and/or mutual authentication.

A Tag that supports the *Untraceable* command shall provide the Tag privileges shown in Table 6.22. A Tag that supports one or more cryptographic suites shall provide the Tag privileges shown in Table 6.23.

The privileges field in a *TagPrivilege* command is 16 bits in length, with a bit for each corresponding Tag privilege. Privileges 12–15 in Table 6.22 and Table 6.23 are assigned by this protocol. Privileges 8–11 are RFU for the access password (Table 6.22); they are assigned by the cryptographic suite for all keys (Table 6.23). Privileges 4–7 are RFU for all keys. Privileges 0–3 are defined by the Tag manufacturer and not specified by this protocol.

The labels in Table 6.22 and Table 6.23 are defined as follows:

— **Privilege Name:** The name of the Tag privilege

— **Bit Assignment:** The bit location in the privileges field for the named privilege, MSB first (i.e. bit 15 is the leading bit in the privileges field in a *TagPrivilege* command and in a Tag's reply).

— **Privilege:** Whether a Tag grants or denies the privilege. A 1_2 means an asserted or granted privilege; a 0_2 means a deasserted or denied privilege.

— **CryptoSuperuser**: Whether a Tag grants the crypto superuser privilege to a key. If **CryptoSuperuser**=1 then a Tag grants the crypto superuser privilege to the key; if **CryptoSuperuser**=0 then a Tag denies the privilege. See below for a description of the crypto superuser privilege.

— **AuthKill**: Whether a Tag grants the authenticated-kill privilege to a key. If **AuthKill**=1 then a Tag grants the authenticated-kill privilege to the key; if **AuthKill**=0 then a Tag denies the privilege.

— **Untraceable**: Whether a Tag executes an *Untraceable* command from, and exposes untraceably hidden memory to, an Interrogator that supplies the access password or key. If **Untraceable**=1 then a Tag grants the privilege; if **Untraceable**=0 then a Tag denies the privilege.

— **DecFilePriv**: Whether a Tag allows an Interrogator that supplies the access password or key the privilege of decrementing file privileges using a *FilePrivilege* command. If **DecFilePriv**=1 in Table 6.22 then a Tag permits an Interrogator that supplies the access password to decrement file privileges for the **open** state and for the access password. If **DecFilePriv**=1 in Table 6.23 then a Tag permits an Interrogator that supplies the key to decrement file privileges for the **open** state and for that key. If **DecFilePriv**=0 then the Tag denies the associated privilege.

— **KeyProperty_N**: One of four key properties (N = 1, 2, 3, 4) defined by the cryptographic suite with which the key is associated.

— **AuthSensorOp**: Whether a Tag with sensor support allows an Interrogator that supplies the access password or key the privilege to execute secure operations (i.e. by an authenticated Interrogator) on the Tag's sensor and/or sensor data. If AuthSensorOp=1, then the Tag grants the privilege. If AuthSensorOp=0, then the Tag denies the privilege.

— **Custom**: One of four key properties (N = 1, 2, 3, 4) defined by the Tag manufacturer.

A Tag that implements the *TagPrivilege* command shall permit an Interrogator that authenticated itself as a crypto superuser in a cryptographic suite to:

— change the value of any key in that cryptographic suite, including its own, using a *KeyUpdate*.

— read or modify privileges (value in Table 6.23) for any key in that cryptographic suite, including its own.

A Tag shall not permit an Interrogator that did not authenticate itself as a crypto superuser to:

— change the value of any key other than the one it used to authenticate itself.

— read or modify privileges (value in Table 6.23) for any key other than the one it used to authenticate itself

— assert a deasserted privilege (value in Table 6.23) for the key it used to authenticate itself.

A Tag that supports the *TagPrivilege* command shall permit an Interrogator that supplies the access password (even if zero-valued) or a key to deassert a privilege for the access password or that key, respectively, regardless of the CryptoSuperuser value.

Because only a crypto superuser can assert a deasserted privilege but there is no crypto superuser for the access password, an access-password privilege, once deasserted, cannot be reasserted.

A Tag manufacturer may configure one or more Tag privileges as permanent and unchangeable, in which case these Tag privileges will not be changeable even by a crypto superuser. A Tag that receives a *TagPrivilege* that attempts to change an unchangeable Tag privilege value shall not execute the *TagPrivilege* and instead treat the command's parameters as unsupported (see Table C.30).

If a Tag supports the *TagPrivilege* command then this protocol recommends that the Tag manufacturer provide at least one nonzero-valued key with crypto superuser privileges for each cryptographic suite supported by the Tag.

Table 6.22 — Tag privileges associated with the access password

Privilege Name	Bit Assignment	Privilege
CryptoSuperuser	**15**	0 (unchangeable)
AuthKill	**14**	0 (unchangeable)
Untraceable	**13**	0/1
DecFilePriv	**12**	0/1
RFU	**11**	0
RFU	**10**	0

Table 6.22 *(continued)*

Privilege Name	Bit Assignment	Privilege
RFU	**9**	0
RFU	**8**	0
RFU	**7**	0
RFU	**6**	0
RFU	**5**	0
AuthSensorOp	**4**	0/1
Custom_1	**3**	Defined by Tag manufacturer
Custom_2	**2**	Defined by Tag manufacturer
Custom_3	**1**	Defined by Tag manufacturer
Custom_4	**0**	Defined by Tag manufacturer

A cryptographic suite defines whether and when:

— a Tag considers an Interrogator to be authenticated.

— an Interrogator considers a Tag to be authenticated.

Table 6.23 — Tag privileges associated with a cryptographic suite

Privilege Name	Bit Assignment	Privilege for Key_0 [1]	...	Privilege for Key_N [1]
CryptoSuperuser	**15**	0/1		0/1
AuthKill	**14**	0/1		0/1
Untraceable	**13**	0/1		0/1
DecFilePriv	**12**	0/1		0/1
KeyProperty_1	**11**	Defined by crypto suite		Defined by crypto suite
KeyProperty_2	**10**	Defined by crypto suite		Defined by crypto suite
KeyProperty_3	**9**	Defined by crypto suite		Defined by crypto suite
KeyProperty_4	**8**	Defined by crypto suite		Defined by crypto suite
RFU	**7**	0		0
RFU	**6**	0		0
RFU	**5**	0		0
AuthSensorOp	**4**	0/1		0/1
Custom_1	**3**	Defined by Tag manufacturer		Defined by Tag manufacturer
Custom_2	**2**	Defined by Tag manufacturer		Defined by Tag manufacturer
Custom_3	**1**	Defined by Tag manufacturer		Defined by Tag manufacturer
Custom_4	**0**	Defined by Tag manufacturer		Defined by Tag manufacturer

Note 1 Each key is assigned to one and only one cryptographic suite.

The cryptographic suite also defines:

— cryptographic conditions that cause a Tag to treat a command's parameters as unsupported.

— cryptographic errors that cause a Tag to transition from the **open** or **secured** state to the **arbitrate** state.

After a successful Interrogator authentication a Tag in the **open** state shall transition to the **secured** state. If the Tag was already in the **secured** state then it remains in the **secured** state. The authenticated

Interrogator shall subsequently encapsulate all commands designated “Mandatory Encapsulation” in Table 6.28 in an *AuthComm* or *SecureComm*. If a Tag receives such a command from an authenticated Interrogator without encapsulation then it shall not execute the command and instead treat the command’s parameters as unsupported (see Table C.30).

A Tag shall transition back to the **open** state, reset its cryptographic engine, and revert to **open**-state file privileges (see below) when an authenticated Interrogator loses its authentication. There are many reasons why an Interrogator may lose its authentication, including but not limited to the Tag receiving a security access command with an incorrect handle, the Tag receiving an invalid command, or the Interrogator starting a new authentication. As a consequence of an Interrogator losing its authentication a Tag with a zero-valued access password may be in the **open** state, in which case the Interrogator may issue an *Access*-command sequence with the zero-valued access password to move the Tag back to the **secured** state (but the Interrogator will still not be authenticated — only a successful *Authenticate* command or *Authenticate*-command sequence authenticates an Interrogator).

An unauthenticated Interrogator may issue an *AuthComm* or a *SecureComm* to an authenticated Tag in the **open** or **secured** state. If the Tag was not previously authenticated by a *Challenge* or *Authenticate* command then it shall not execute the command and instead treat the command’s parameters as unsupported (see Table C.30).

If a condition of a cryptographic suite causes a Tag to transition from the **open** or **secured** state to the **arbitrate** state then the Tag (i) shall not change the value of its inventoried flag, and (ii) shall reset its cryptographic engine.

6.3.2.11.3 File-management access commands

The file-management access commands are *FileOpen*, *FileList*, *FileSetup*, and *FilePrivilege*. All are optional. A Tag that supports File_N, N>0 shall implement *FileOpen*; it may implement *FileList*, *FileSetup, and FilePrivilege* as well. A Tag may implement one or more of these commands regardless of whether the Tag supports optional core commands and/or cryptographic security.

A Tag may implement zero, one, or more than one file in User memory. If a Tag implements a single file then that file shall be File_0. A Tag with User memory shall open File_0 upon first entering the **open** or **secured** state. If a Tag implements multiple files then it may subsequently close File_0 and open another file. A Tag shall have only a single file open at any time. All access commands operate on the currently open file.

Each file shall have an 8-bit FileType and a 10-bit FileNum unless a Tag does not support any file-management access commands, in which case a Tag that implements File_0 may omit FileType and FileNum.

— A Tag manufacturer shall preassign a FileType to each file supported by the Tag. If a Tag does not support the *FileSetup* command then FileType is not changeable in the field and all files have FileType=00_h. If a Tag supports the *FileSetup* command then FileType is changeable in the field and FileType=00_h indicates that a file’s type is currently unassigned.

— A Tag manufacturer shall preassign a unique FileNum to each file supported by the Tag. FileNum is not changeable in the field. A Tag may support up to 1023 files, numbered 0 to 1022 (0000000000_2 – 1111111102_2). The files may have different size (including zero size). FileNum=0000000000_2 shall be reserved for the base file (File_0) of User memory. FileNum=1111111111_2 shall be RFU. This protocol recommends, but does not require, that Tag manufacturers number files sequentially.

A *FileOpen* command allows an Interrogator to open a file. Upon receiving a *FileOpen* a Tag shall first close the currently open file and then open the new file, with the new file’s starting address mapped to 00_h of User memory. An Interrogator may be able to subsequently read, write, erase, blockpermalock, resize, or modify privileges for the newly opened file depending on the Tag state and if/how the Interrogator authenticated itself.

A *FileList* command allows an Interrogator to determine the existence of, size of, attributes of, and its privileges to, one or more files.

A *FileSetup* command allows an Interrogator to change the FileType for, and/or resize, the currently open file. Only a *dynamic* Tag (see below) is capable of resizing a file.

A *FilePrivilege* command allows an Interrogator to read or alter the privileges (see below) granted by the currently open file to the **open** state, access password, or a key.

A Tag manufacturer shall precreate all files; the number of files shall not be changeable in the field. This protocol defines two types of Tags, *static* and *dynamic*, according to their memory-allocation features as follows:

Static: A manufacturer of a *static* Tag shall preallocate all User memory to files. A *static* Tag may permit changing a file's FileType but shall not permit file resizing.

Dynamic: A manufacturer of a *dynamic* Tag may preallocate no, some, or all User memory to files. A *dynamic* Tag may permit file resizing by an Interrogator that has a file superuser privilege.

A Tag manufacturer shall decide where a Tag stores its FileType and FileNum data and may choose a readable portion of memory (if desired). Regardless of the location, a Tag shall not allow an Interrogator to modify a file's type by any command except *FileSetup*, and shall not allow an Interrogator to modify a FileNum by any means.

Files may range in size from a minimum of zero to a maximum of 1022 blocks. Commands that include a FileSize parameter use 10 bits to specify sizes from zero to 1022 blocks (0000000000_2 – 1111111110_2, respectively). FileSize 1111111111_2 shall be RFU.

Block size may be one to 1024 words. A Tag manufacturer shall predefine a single fixed, unchangeable block size that the Tag shall use for all file allocation as well as for the *BlockPermalock* command. Tag manufacturers shall not use block sizes exceeding 1024 words. Tag replies that return a BlockSize value use 10 bits to specify the size from one (0000000000_2) to 1024 (1111111111_2) words. BlockSize does not have an RFU value.

This protocol allows file sizes from 0 to 16,744,448 bits (max FileSize=1022 blocks, max BlockSize=1024 words, word=16 bits).

If a Tag supports File_0 then it shall provide the file privileges shown in Table 6.24. If a Tag supports File_N, N>0 then it shall also provide the file privileges shown in Table 6.25. Each file has a 4-bit privilege for the **open** state, for the access password in the **secured** state, and for each key in the **secured** state, as follows:

Open state: Each file has a 4-bit **open**-state privilege. A Tag with M files shall implement M 4-bit **open**-state file privileges, one for each file.

Access password (secured state): Each file has a single 4-bit privilege for the access password (even if the access password is zero-valued). A Tag with M files shall implement M 4-bit **secured**-state access-password file privileges, one for each file.

Key (secured state): Each file has a 4-bit privilege for each key implemented by the Tag. A Tag with M files and N keys shall implement M×N 4-bit **secured**-state key file privileges.

Given the above, a Tag with N keys and M files supports (2+N)×M independent 4-bit privileges. For example, suppose a Tag implements N=2 keys and File_0, File_1, and File_2. Then the Tag has 4×3=12 4-bit privileges.

A *FilePrivilege* may assign privileges 0000_2–0011_2 and 1100_2–1111_2 in Table 6.24 and Table 6.25. If a Tag does not implement any manufacturer-defined privileges then the Tag may store only the 2 LSBs of the 4-bit privilege, but all communications still use 4-bit values. In this latter case, if an Interrogator sends a 4-bit privilege with either MSB being nonzero then the Tag shall not execute the *FilePrivilege* and instead treat the command's parameters as unsupported (see Table C.30).

The access password or a key with a 0011_2 **secured**-state file privilege in Table 6.24 or Table 6.25 is defined to be a superuser for that file.

A Tag shall permit an Interrogator that accessed or authenticated itself as a file superuser to:

— read or assign a new 4-bit privilege for the **open** state, access password, or any key (including its own) regardless of the cryptographic suite to which the key is assigned, for the currently open file, using a *FilePrivilege* command.

— change the FileType of the currently open file using a *FileSetup* command, for a *static* or a *dynamic* Tag.

— resize the currently open file using a *FileSetup* command, but only if the file contains no permalocked or permaunlocked memory and only if the Tag is *dynamic*.

A Tag shall not permit an Interrogator that did not access or authenticate itself as a file superuser to:

— read or assign the 4-bit privilege for the **open** state, for the currently open file.

— read or assign the 4-bit privilege for the access password or for any key other than the one it used to enter the **secured** state, for the currently open file.

— increase the privileges (move down one or more rows in Table 6.24 or Table 6.25) for the access password or for any key, for the currently open file.

If the access password or key that a Tag used to enter the **secured** state has DecFilePriv=1 (see Table 6.22 and Table 6.23) then a Tag shall permit an Interrogator to self-reduce its privileges (move up one or more rows in Table 6.24 or Table 6.25) to the currently open file for this access password or key. To be clear, if DecFilePriv=1 then an Interrogator may, via a *FilePrivilege* command, instruct a Tag to decrement a 0011_2 privilege to 0010_2, 0001_2, or 0000_2; a 0010_2 privilege to 0001_2 or 0000_2; or a 0001_2 privilege to 0000_2 for the currently open file for the access password or key that the Tag used to enter the **secured** state.

Table 6.24 — File_0 privileges

Privilege value	Open State (Privilege by File)					Privilege value	Secured State (Privilege by Access Password or Key)				
	Read	Write BlockWrite BlockErase	Lock Block-Permalock	File-Privilege	File-Setup		Read	Write BlockWrite BlockErase	Lock Block-Permalock	File-Privilege	File-Setup
0000	✓	×	D	D	D	0000	✓	×	×	×	×
0001	✓	×	D	D	D	0001	✓	L	×	P	×
0010	✓	L	D	D	D	0010	✓	L	✓	P	×
0011	✓	L	D	D	D	0011 [1]	✓	L	✓	✓	✓
0100	RFU					0100	RFU				
0101	RFU					0101	RFU				
0110	RFU					0110	RFU				
0111	RFU					0111	RFU				
1000	RFU					1000	RFU				
1001	RFU					1001	RFU				
1010	RFU					1010	RFU				
1011	RFU					1011	RFU				
1100	Manufacturer defined					1100	Manufacturer defined				
1101	Manufacturer defined					1101	Manufacturer defined				
1110	Manufacturer defined					1110	Manufacturer defined				

Key: ✓=allowed by privilege; ×=disallowed by privilege; L=allowance determined by lock and blockpermalock status of the specified memory banks/blocks; P=allowance determined by DecFilePriv for the password or key; D=command not permitted in the state.

Table 6.24 *(continued)*

Privilege value	Open State (Privilege by File)					Privilege value	Secured State (Privilege by Access Password or Key)				
	Read	Write BlockWrite BlockErase	Lock Block-Perma lock	File-Privi-lege	File-Setup		Read	Write BlockWrite BlockErase	Lock Block-Perma lock	File-Privi-lege	File-Setup
1111	Manufacturer defined					1111	Manufacturer defined				
Key: ✓=allowed by privilege; ×=disallowed by privilege; L=allowance determined by lock and blockpermalock status of the specified memory banks/blocks; P=allowance determined by DecFilePriv for the password or key; D=command not permitted in the state.											

Note 1 This 0011 **secured**-state privilege is a superuser for File_0.

Table 6.25 — File_N (N>0) privileges

Privilege value	Open State (Privilege by File)					Privilege value	Secured State (Privilege by Access Password or Key)				
	Read	Write BlockWrite BlockErase	Block-Perma lock	File-Privi-lege	File-Setup		Read	Write BlockWrite BlockErase	Block-Perma lock	File-Privi-lege	File-Setup
0000	×	×	D	D	D	0000	×	×	×	×	×
0001	✓	×	D	D	D	0001	✓	×	×	P	×
0010	✓	L	D	D	D	0010	✓	L	×	P	×
0011	✓	L	D	D	D	0011 [1]	✓	L	✓	✓	✓
0100	RFU					0100	RFU				
0101	RFU					0101	RFU				
0110	RFU					0110	RFU				
0111	RFU					0111	RFU				
1000	RFU					1000	RFU				
1001	RFU					1001	RFU				
1010	RFU					1010	RFU				
1011	RFU					1011	RFU				
1100	Manufacturer defined					1100	Manufacturer defined				
1101	Manufacturer defined					1101	Manufacturer defined				
1110	Manufacturer defined					1110	Manufacturer defined				
1111	Manufacturer defined					1111	Manufacturer defined				
Key: ✓=allowed by privilege; ×=disallowed by privilege; L=allowance determined by the blockpermalock status of the specified memory blocks; P=allowance determined by DecFilePriv for the password or key; D=command not permitted in the state.											

Note 1 This 0011 **secured**-state privilege is a superuser for File_N, N>0.

Table 6.26 — Allowed file resizing

File number	File is permalocked or permaunlocked	One or more file blocks are permalocked or permaunlocked	Increasing file size allowed?	Decreasing file size allowed?
N = 0	No	No	Yes	Yes
	No	Yes	Yes	No
	Yes	No	No	No
	Yes	Yes	No	No
N > 0	N/A	No	Yes	Yes
		Yes	Yes	No

Table 6.27 — Access commands and Tag states in which they are permitted

Command	State			Subclass	Remark
	Acknowledged	Open	Secured		
Req_RN	allowed	allowed	allowed	Core	mandatory command
Read	disallowed	allowed	allowed	Core	mandatory command
Write	disallowed	allowed	allowed	Core	mandatory command; requires prior *Req_RN*
Kill (password-based)	disallowed	allowed	allowed	Core	mandatory command; requires prior *Req_RN*
Kill (authenticated)	disallowed	disallowed	allowed	Core	optional usage of mandatory *Kill* command
Lock	disallowed	disallowed	allowed	Core	mandatory command
Access	disallowed	allowed	allowed	Core	optional command; requires prior *Req_RN*
BlockWrite	disallowed	allowed	allowed	Core	optional command
BlockErase	disallowed	allowed	allowed	Core	optional command
BlockPermalock	disallowed	disallowed	allowed	Core	optional command
ReadBuffer	disallowed	allowed	allowed	Core	optional command
Untraceable	disallowed	disallowed	allowed	Core	optional command
Authenticate	disallowed	allowed	allowed	Security	optional command
AuthComm	disallowed	allowed	allowed	Security	optional command; requires prior authentication
SecureComm	disallowed	allowed	allowed	Security	optional command; requires prior authentication
KeyUpdate	disallowed	disallowed	allowed	Security	optional command; requires prior authentication
TagPrivilege	disallowed	disallowed	allowed	Security	optional command
FileOpen	disallowed	allowed	allowed	File	optional command
FileList	disallowed	allowed	allowed	File	optional command
FilePrivilege	disallowed	disallowed	allowed	File	optional command
FileSetup	disallowed	disallowed	allowed	File	optional command

A Tag manufacturer may assign file privileges to the **open** state and to the access password or keys as required by the Tag's intended use case. A Tag manufacturer may assign file superuser privileges

to the access password or to any key. Although this protocol recommends against a Tag manufacturer assigning file superuser privileges to a zero-valued access password or key, it does not prohibit a Tag manufacturer from doing so.

As described above, a Tag opens File_0 upon first entering the **open** or **secured** state. If an Interrogator attempts to subsequently open another file for which its privilege level is 0000_2 then the Tag opens the file but does not grant the Interrogator any file privileges.

Some privilege fields in Table 6.24 and Table 6.25 show "×" (disallowed by privilege). If *Read* is "×" for a privilege value then a Tag shall behave as if the memory location does not exist. Otherwise, if *Write*, *BlockWrite*, or *BlockErase* are "×" then the Tag shall behave as if the memory location is permalocked; and if *Lock* or *BlockPermalock* are "×" then the Tag shall behave as if the memory location is neither lockable nor unlockable. If *FilePrivilege* or *FileSetup* are "×" then the Tag shall behave as if the Interrogator has insufficient privileges. If a Tag implements the *BlockPermalock* command then all files shall support the *BlockPermalock* command.

If a Tag's User memory is untraceably hidden then the Tag shall only execute a *FileOpen*, *FileList*, *FileSetup*, or *FilePrivilege* issued by an Interrogator with an asserted Untraceable privilege (see Table 6.22 and Table 6.23); if the Interrogator has a deasserted Untraceable privilege then the Tag shall treat these commands' parameters as unsupported (see Table C.30).

A Tag shall not permit a permalocked portion of memory to be erased or overwritten, except for the **L** and **U** bits in UII memory, which an Interrogator with an asserted Untraceable privilege may overwrite.

In some instances a *dynamic* Tag may allow file resizing. Whether a Tag allows resizing shall depend on whether the Tag accepts a *FileSetup* command (varies by privilege and state), whether the Tag has free memory available for the resizing, and whether the file or any blocks in it are permalocked or permaunlocked. See Table 6.26.

6.3.2.12 Interrogator commands and Tag replies

Interrogator-to-Tag commands shall use the command codes, protection, and parameters shown in Table 6.28.

— *QueryRep* and *ACK* have 2-bit command codes beginning with 0_2.

— *Query*, *QueryAdjust*, and *Select* have 4-bit command codes beginning with 10_2.

— Other commands that are sensitive to link throughput use 8-bit command codes beginning with 110_2.

— Other commands that are insensitive to link throughput use 16-bit command codes beginning with 1110_2.

— *QueryRep*, *ACK*, *Query*, *QueryAdjust,* and *NAK* have the unique command lengths shown in Table 6.28. No other commands shall have these lengths. If a Tag receives one of these commands with an incorrect length then it shall treat the command as invalid (see Table C.30).

— *Query* and *Flex_Query* (see 7.4.1) are protected by a CRC-5, shown in Table 6.12 and detailed in Annex F.

— *Select*, *Req_RN*, *Read*, *Write*, *Kill*, *Lock*, *Access*, *BlockWrite*, *BlockErase*, *BlockPermalock*, *Authenticate*, *SecureComm*, *AuthComm*, *KeyUpdate*, *ReadBuffer*, *Challenge*, *Untraceable*, *FileOpen, FileList, FilePrivilege, FileSetup*, *TagPrivilege*, *BroadcastSync* (see 8.3.3), and *HandleSensor* (see 8.4) are protected by a CRC-16, defined in 6.3.1.5 and detailed in Annex F.

— R=>T commands begin with either a preamble or a frame-sync, as described in 6.3.1.2.8. The command-code lengths specified in Table 6.28 do not include the preamble or frame-sync.

— A Tag's behavior upon receiving a faulty command depends on the fault type and the Tag state. Annex B and Annex C define the fault types by state. In general, the faults are (1) unsupported parameters, (2) incorrect handle, (3) improper, and (4) invalid. Note that, for some cryptographic suites, a Tag may reset its cryptographic engine and change state upon receiving a faulty command.

Table 6.28 — Interrogator Commands

Command	Code	Length (bits)	Mandatory Command (Y/N)?	Reply Type	Encapsulation			Protection
					SecureComm[2] (Y/N)?	AuthComm[2] (Y/N)?	Mandatory[3] (Y/N)?	
QueryRep	**00**	4	Yes	Immediate	No	No	No	Unique length
ACK	**01**	18	Yes	Immediate	Yes	Yes	No	Unique length
Query	**1000**	22	Yes	Immediate	No	No	No	Unique length and a CRC-5
QueryAdjust	**1001**	9	Yes	Immediate	No	No	No	Unique length
Select	**1010**	> 44	Yes	None	No	No	No	CRC-16
Reserved for future use	**1011**	–	–	–	–	–	–	–
NAK	**11000000**	8	Yes	Immediate	No	No	No	Unique length
Req_RN	**11000001**	40	Yes	Immediate	Yes	Yes	No	CRC-16
Read	**11000010**	> 57	Yes	Immediate	Yes	Yes	Yes	CRC-16
Write	**11000011**	> 58	Yes	Delayed	No	No	No	CRC-16
Kill	**11000100**	59	Yes	Delayed	Yes [1]	Yes [1]	Yes [1]	CRC-16
Lock	**11000101**	60	Yes	Delayed	Yes	Yes	Yes	CRC-16
Access	**11000110**	56	No	Immediate	No	No	No	CRC-16
BlockWrite	**11000111**	> 57	No	Delayed	Yes	Yes	Yes	CRC-16
BlockErase	**11001000**	> 57	No	Delayed	Yes	Yes	Yes	CRC-16
BlockPermalock	**11001001**	> 66	No	Immediate & Delayed	Yes	Yes	Yes	CRC-16
Reserved for BAP Manchester (see 7.5.4)	**11001010** … **11001110**	–	–	–	–	–	–	–
Flex_Query (see 7.4.1)	**11001111**	42	No	Immediate	No	No	No	Unique length and a CRC-5
Reserved for BAP Manchester (see7.5.4)	**11010000**	–	–	–	–	–	–	–
BroadcastSync (see 8.3.3)	**11010001**	56	No	None	No	No	No	CRC-16
ReadBuffer	**11010010**	67	No	Immediate	No	Yes	No	CRC-16
FileOpen	**11010011**	52	No [4]	Immediate	Yes	Yes	Yes	CRC-16
Challenge	**11010100**	> 48	No	None	No	No	No	CRC-16
Authenticate	**11010101**	> 64	No	In-process	No	No	No	CRC-16
SecureComm	**11010110**	> 56	No	In-process	No	No	No	CRC-16
AuthComm	**11010111**	> 42	No	In-process	No	No	No	CRC-16
Reserved for future use	**11011000**	–	–	–	–	–	–	–
HandleSensor (see 8.4)	**11011001**	> 63	No	Delayed	Yes	Yes	Yes	CRC-16

Table 6.28 *(continued)*

Command	Code	Length (bits)	Mandatory Command (Y/N)?	Reply Type	Encapsulation			Protection
					SecureComm[2] (Y/N)?	AuthComm[2] (Y/N)?	Mandatory[3] (Y/N)?	
Reserved for future use	**110**11010 ... **110**11111	–	–	–	–	–	–	–
Reserved for custom commands	**11100000** 00000000 ... **11100000** 11111111	–	–	–	–	–	–	Manufacturer defined
Reserved for proprietary commands	**11100001** 00000000 ... **11100001** 11111111	–	–	–	–	–	–	Manufacturer defined
Untraceable	**11100010** 00000000	62	No	Delayed	Yes	Yes	Yes	CRC-16
FileList	**11100010** 00000001	71	No	In-process	Yes	Yes	Yes	CRC-16
KeyUpdate	**11100010** 00000010	> 72	No	In-process	Yes	Yes	No	CRC-16
TagPrivilege	**11100010** 00000011	78	No	In-process	Yes	Yes	Yes	CRC-16
FilePrivilege	**11100010** 00000100	68	No	In-process	Yes	Yes	Yes	CRC-16
FileSetup	**11100010** 00000101	71	No	In-process	Yes	Yes	Yes	CRC-16
Reserved for future use	**11100010** 00000110 ... **11101111** 11111111	–	–	–	–	–	–	–

Note 1 An authenticated *Kill* shall be encapsulated in a *SecureComm* or an *AuthComm*; a password-based *Kill* shall not be encapsulated.

Note 2 Commands with a "yes" may be encapsulated in a *SecureComm* or an *AuthComm*, as appropriate.

Note 3 For an authenticated Interrogator and commands with a "yes", encapsulation in a *SecureComm* or an *AuthComm* is mandatory.

Note 4 If a Tag supports File_N, N>0 then *FileOpen* is mandatory.

6.3.2.12.1 Select commands

The select command set comprises *Select* and *Challenge*.

6.3.2.12.1.1 *Select* (mandatory)

Interrogators and Tags shall implement the *Select* command shown in Table 6.29. A *Select* allows an Interrogator to select a Tag subpopulation based on user-defined criteria, enabling union (U), intersection (∩), and negation (~) based Tag partitioning. Interrogators perform U and ∩ operations by issuing successive *Select* commands. *Select* can assert or deassert a Tag's **SL** flag, which applies across

all four sessions, or it can set a Tag's **inventoried** flag to either *A* or *B* in any one of the four sessions. A Tag executes a *Select* from any state except **killed**. *Select* passes the following parameters from Interrogator to Tags:

— Target indicates whether the *Select* modifies a Tag's **SL** flag or its **inventoried** flag, and in the case of **inventoried** it further specifies one of four sessions. A *Select* that modifies the **SL** flag shall not modify an **inventoried** flag, and vice versa. A Tag shall ignore a *Select* whose Target is 101_2, 110_2, or 111_2.

— Action elicits the Tag behavior in Table 6.30, in which matching and not-matching Tags assert or deassert **SL** or set their **inventoried** flag to *A* or *B*. A Tag conforming to the contents of the MemBank, Pointer, Length, and Mask fields is matching. A Tag not conforming to the contents of these fields is not-matching. The criteria for determining whether a Tag is matching or not-matching are specified by the MemBank, Pointer, Length and Mask fields.

— MemBank specifies how a Tag applies Mask. If MemBank=00_2 then the Tag searches for at least one file whose FileType matches Mask. If MemBank=01_2, 10_2, 11_2 then a Tag applies Mask to the UII memory bank, TID memory bank, or File_0, respectively. A *Select* specifies a single FileType or memory bank. Successive *Selects* may apply to different file types and/or memory banks.

— Pointer specifies a starting bit address for the Mask comparison. Pointer uses EBV formatting (see Annex A) and bit (not word) addressing. If MemBank=00_2 then an Interrogator shall set Pointer to 00_h; if a Tag receives a *Select* with MemBank=00_2 and a nonzero Pointer value then it shall ignore the *Select*.

— Length specifies the length of Mask. Length is 8 bits, allowing Masks from 0 to 255 bits in length. If MemBank=00_2 then an Interrogator shall set Length=00001000_2; if a Tag receives a *Select* with MemBank=00_2 and Length<>00001000_2 then it shall ignore the *Select*.

— Mask is either a FileType (if MemBank=00_2) or a bit string that a Tag compares to a memory location that begins at Pointer and ends Length bits later (if MemBank<>00_2). An untraceable Tag shall process a *Select* with MemBank=00_2 whose User memory is traceable, or with MemBank<>00_2 whose Mask operates on a completely traceable bit string. A Tag shall treat as not-matching a *Select* command whose Mask includes untraceably hidden memory.

  — MemBank=00_2: If a Tag has a file with the specified FileType then the Tag is matching. If the Tag does not support files or does not have a file with the specified FileType then the Tag is not-matching.

  — MemBank<>00_2: If Mask matches the string specified by Pointer and Length then the Tag is matching. If Pointer and Length reference a memory location that does not exist then the Tag is not-matching. If Length is zero then the Tag is matching, unless Pointer references a memory location that does not exist, or Truncate=1 and Pointer is outside the UII specified in the length field in the StoredPC, in which case the Tag is not-matching.

— Truncate indicates whether a Tag's backscattered reply shall be truncated to those UII bits that follow Mask. If an Interrogator asserts Truncate, and if a subsequent *Query* specifies Sel=10 or Sel=11, then a matching Tag shall truncate its *ACK* reply to the portion of the UII immediately following Mask, followed by a PacketCRC. If an Interrogator asserts Truncate then it shall assert it:

  — in the last *Select* that the Interrogator issues prior to sending a *Query*,

  — only if the *Select* has Target=100_2, and

  — only if Mask ends in the UII.

These constraints *do not* preclude an Interrogator from issuing multiple *Select* commands that target the **SL** and/or **inventoried** flags. They *do* require that an Interrogator that is requesting Tags to truncate their replies assert Truncate in the last *Select*, and that this last *Select* targets the **SL** flag. A Tag shall decide whether to truncate its backscattered UII on the basis of the most recently received valid *Select* (i.e. not ignored and matching or not-matching).

If a Tag receives a *Select* with Truncate=1 and

— Target<>100_2 or MemBank<>01_2 then the Tag shall ignore the *Select*.

— MemBank=01_2 but Mask ends outside the UII specified by the **L** bits in the StoredPC then the Tag shall be not-matching.

A Tag shall preface a truncated reply with five leading zeros (00000_2) inserted between the preamble and the truncated reply. Specifically, when truncating its replies a Tag backscatters 00000_2, then the portion of its UII following Mask, and then a PacketCRC. See Table 6.17.

A Tag shall power-up with Truncate=0.

Mask may end at the last bit of the UII, in which case a truncating Tag shall backscatter 00000_2 followed by a PacketCRC.

Truncated replies never include an XPC_W1 or an XPC_W2, because Mask must end in the UII.

Table 6.29 — *Select* command

	Command	Target	Action	MemBank	Pointer	Length	Mask	Truncate	CRC
# of bits	4	3	3	2	EBV	8	Variable	1	16
description	1010	000: **Inventoried** (S0) 001: **Inventoried** (S1) 010: **Inventoried** (S2) 011: **Inventoried** (S3) 100: **SL** 101: RFU 110: RFU 111: RFU	See Table 6.30	00: FileType 01: UII 10: TID 11: File_0	Starting Mask address	Mask length (bits)	Mask value	0: Disable truncation 1: Enable truncation	CRC-16

Because a Tag stores its StoredPC and StoredCRC in UII memory, a *Select* command may select on them. Because a Tag computes its PacketPC and PacketCRC dynamically and does not store them in memory, a *Select* command is unable to select on them.

Table 6.30 — Tag response to Action parameter

Action	Tag Matching	Tag Not-Matching
000	assert **SL** or **inventoried** → *A*	deassert **SL** or **inventoried** → *B*
001	assert **SL** or **inventoried** → *A*	do nothing
010	do nothing	deassert **SL** or **inventoried** → *B*
011	negate **SL** or (*A* → *B*, *B* → *A*)	do nothing
100	deassert **SL** or **inventoried** → *B*	assert **SL** or **inventoried** → *A*
101	deassert **SL** or **inventoried** → *B*	do nothing
110	do nothing	assert **SL** or **inventoried** → *A*
111	do nothing	negate **SL** or (*A* → *B*, *B* → *A*)

A *Select* whose Pointer, Length, and Mask include the StoredPC may produce unexpected behavior. Specifically, if a Tag's *ACK* reply uses a PacketPC then the reply may appear to not match Mask even though the Tag's behavior indicates matching, and vice versa. For example, suppose that an Interrogator sends a *Select* to match a 00100_2 length field in the StoredPC. Further assume that the Tag is matching,

but has an asserted **XI**. The Tag will increment its length field to 00101_2 when replying to the *ACK*. The Tag was matching, but the backscattered length field in the PacketPC appears to be not-matching.

An Interrogator shall prepend a *Select* command with a frame-sync (see 6.3.1.2.8). The CRC-16 that protects a *Select* is calculated over the first command-code bit to the Truncate bit.

A Tag shall not reply to a *Select*.

6.3.2.12.1.2 *Challenge* (optional)

Interrogators and Tags may implement the *Challenge* command; if they do then they shall implement it as shown in Table 6.31. A *Challenge* allows an Interrogator to instruct multiple Tags to simultaneously yet independently precompute and store a cryptographic value or values for use in a subsequent authentication. The generic nature of the *Challenge* command allows it to support a wide variety of cryptographic suites. A Tag executes a *Challenge* from any state except **killed**. *Challenge* has the following fields:

— IncRepLen specifies whether the Tag omits or includes length in its stored reply. If IncRepLen=0 then the Tag omits length from its stored reply; if IncRepLen=1 then the Tag includes length in its stored reply.

— Immed specifies whether a Tag concatenates response to its UII when replying to an *ACK*. If immed=0 then the Tag does not concatenate response to its UII when replying to an *ACK*; if immed=1 then the Tag backscatters UII + response when replying to an *ACK*.

— CSI selects the cryptographic suite that Tag and Interrogator use for the *Challenge*.

— Length is the message length in bits.

— Message includes parameters for the authentication.

Upon receiving a *Challenge* a Tag that supports the command shall return to the **ready** state and deassert its **C** flag. If the Tag supports the CSI and can execute message then it shall perform the requested action(s); otherwise the Tag shall not execute message. A Tag shall not reply to a *Challenge*.

A *Challenge* contains 2 RFU bits. An Interrogator shall set these bits to 00_2. If a Tag receives a *Challenge* containing nonzero RFU bits then it shall return to the **ready** state and deassert its **C** flag but not execute message. Future protocols may use these RFU bits to expand the functionality of the *Challenge* command.

An Interrogator shall prepend a *Challenge* command with a frame-sync (see 6.3.1.2.8). The CRC-16 that protects a *Challenge* is calculated over the first command-code bit to the last Message bit.

If a Tag supports the *Challenge* command then it shall implement the security (**S**) indicator (see 6.3.2.1.3).

The cryptographic suite specifies message formatting, what value or values the Tag precomputes, and the format in which the Tag stores the value(s). It specifies Tag behavior if a Tag cannot compute one or more values. It may contain additional information such as how Tag and Interrogator perform a subsequent authentication from values precomputed by a *Challenge*. It specifies the formatting of the computed result for both a successful and an unsuccessful *Challenge*. It may contain information about how Tag and Interrogator derive session keys for subsequent communications. It may include parameters, such as a key, that affect pre- and post-authenticated communications. See Annex M for the parameters specified by a cryptographic suite.

After executing a *Challenge* a Tag shall store its response (result or error code) in its ResponseBuffer. If IncRepLen=1 then the Tag also stores the length of the response (in bits) as shown in Figure 6.17. An Interrogator may subsequently read the response using a *ReadBuffer* command.

After executing and storing a response a Tag shall assert its **C** flag. A Tag shall not assert its **C** flag until after it has computed and stored the entire response. A Tag shall deassert its **C** flag upon (a) receiving a subsequent *Challenge*, or (b) exceeding the **C** flag persistence time in Table 6.20. As described above the

ResponseBuffer contents may include a length field and may be a cryptographic response or an error code. A Tag does not permit an Interrogator to read its ResponseBuffer when **C**=0.

If the most recent *Challenge* received and executable by a Tag asserts immed, and if the Tag's **C** flag is asserted when it receives a subsequent *ACK*, then when replying to the *ACK* the Tag shall concatenate its ResponseBuffer contents to its UII and backscatter the concatenated reply. See Table 6.17. See also Figure 6.23.

A *Challenge* may precede a *Query*. Tags that hear a *Challenge* and support the command, the CSI, and message compute their results simultaneously. An Interrogator may select on the **C** flag to preferentially inventory Tags that have successfully stored a response.

If an Interrogator sends a command while a Tag is processing a *Challenge* then the Tag may abort its processing (leaving **C**=0) and evaluate the command or, in environments with limited power availability, may undergo a power-on reset. This protocol recommends that Interrogators send CW for a sufficient period of time after sending a *Challenge* for all Tags to compute and store their result and assert their **C** flag.

If a Tag observes a properly formatted *Challenge* but there is a cryptographic error, and the cryptographic suite specifies that the error requires a security timeout, then the Tag shall return to **ready** and enforce a security timeout as specified in 6.3.2.5. If a Tag that supports security timeouts for a *Challenge* receives a *Challenge* during a timeout then it shall return to **ready** but not act on or otherwise execute any portion of the *Challenge*.

Table 6.31 — *Challenge* Command

	Command	RFU	IncRepLen	Immed	CSI	Length	Message	CRC
# of bits	8	2	1	1	8	12	Variable	16
description	11010100	00	0: Omit length from reply 1 Include length in reply	0: Do not transmit result with UII 1: Transmit result with UII	CSI	length of message	message (depends on CSI)	CRC-16

6.3.2.12.2 Inventory commands

The inventory command set comprises *Query*, *QueryAdjust, QueryRep, ACK*, and *NAK*.

6.3.2.12.2.1 *Query* (mandatory)

Interrogators and Tags shall implement the *Query* command shown in Table 6.32. *Query* initiates and specifies an inventory round. *Query* includes the following fields:

— DR (TRcal divide ratio) sets the T=>R link frequency as described in 6.3.1.2.8 and Table 6.9.

— M (cycles per symbol) sets the T=>R data rate and modulation format as shown in Table 6.10.

— TRext chooses whether a Tag prepends the T=>R preamble with a pilot tone as described in 6.3.1.3.2.2 and 6.3.1.3.2.4. A Tag's reply to a command that uses a *delayed* or an *in-process* reply (see 6.3.1.6) always uses an extended preamble regardless of the TRext value.

— Sel chooses which Tags respond to the *Query* (see 6.3.2.12.2.1 and 6.3.2.8).

— Session chooses a session for the inventory round (see 6.3.2.8).

— Target selects whether Tags whose **inventoried** flag is *A* or *B* participate in the inventory round. Tags may change their inventoried flag from *A* to *B* (or vice versa) as a result of being singulated.

— Q sets the number of slots in the round (see 6.3.2.8).

An Interrogator shall prepend a *Query* with a preamble (see 6.3.1.2.8).

Table 6.32 — *Query* command

	Command	DR	M	TRext	Sel	Session	Tar-get	Q	CRC
# of bits	4	1	2	1	2	2	1	4	5
description	1000	0: DR=8 1: DR=64/3	00: M=1 01: M=2 10: M=4 11: M=8	0: No pilot tone 1: Use pilot tone	00: All 01: All 10: ~**SL** 11: **SL**	00: S0 01: S1 10: S2 11: S3	0: *A* 1: *B*	0–15	CRC-5

Table 6.33 — Tag reply to a *Query* command

	Reply
# of bits	16
description	RN16

An Interrogator shall not encapsulate a *Query* in a *SecureComm* or *AuthComm* (see Table 6.28).

The CRC-5 that protects a *Query* is calculated over the first command-code bit to the last Q bit. If a Tag receives a *Query* with a CRC-5 error then it shall treat the command as invalid (see Table C.30).

Upon receiving a *Query*, Tags with matching Sel and Target shall pick a random value in the range (0, 2^Q–1), inclusive, and shall load this value into their slot counter. If a Tag, in response to the *Query*, loads its slot counter with zero, then its reply to a *Query* shall be as shown in

Table 6.33 using the *immediate* reply type specified in 6.3.1.6.1; otherwise the Tag shall remain silent.

A *Query* may initiate an inventory round in a new session or in the prior session. If a Tag in the **acknowledged**, **open**, or **secured** states receives a *Query* whose session parameter matches the prior session it shall invert its **inventoried** flag (i.e. *A*→*B* or *B*→*A*) for the session before it evaluates whether to transition to **ready**, **arbitrate**, or **reply**. If a Tag in the **acknowledged**, **open**, or **secured** states receives a *Query* whose session parameter does not match the prior session it shall leave its **inventoried** flag for the prior session unchanged when beginning the new round.

A Tag shall support all DR and M values specified in Table 6.9 and Table 6.10, respectively.

A Tag in any state other than **killed** shall execute a *Query* command, starting a new round in the specified session and transitioning to **ready**, **arbitrate**, or **reply**, as appropriate (see Figure 6.21). A Tag in the **killed** state shall ignore a *Query*.

6.3.2.12.2.2 *QueryAdjust* (mandatory)

Interrogators and Tags shall implement the *QueryAdjust* command shown in Table 6.34. *QueryAdjust* adjusts *Q* (i.e. the number of slots in an inventory round – see 6.3.2.8) without changing any other round parameters.

QueryAdjust includes the following fields:

— Session corroborates the session number for the inventory round (see 6.3.2.8 and 6.3.2.12.2.1). If a Tag receives a *QueryAdjust* whose session number is different from the session number in the *Query* that initiated the round it shall ignore the command.

— UpDn determines whether and how the Tag adjusts *Q*, as follows:

110: Increment *Q* (i.e. *Q* = *Q* + 1).

000: No change to *Q*.

011: Decrement *Q* (i.e. *Q* = *Q* – 1).

If a Tag receives a *QueryAdjust* with an UpDn value different from those specified above then it shall treat the command as invalid (see Table C.30). If a Tag whose *Q* value is 15 receives a *QueryAdjust* with UpDn=110 then it shall change UpDn to 000 prior to executing the command; likewise, if a Tag whose *Q* value is 0 receives a *QueryAdjust* with UpDn=011 then it shall change UpDn to 000 prior to executing the command.

A Tag shall maintain a running count of the current *Q* value. The initial *Q* value is specified in the *Query* command that started the inventory round; one or more subsequent *QueryAdjust* commands may modify *Q*.

An Interrogator shall prepend a *QueryAdjust* with a frame-sync (see 6.3.1.2.8).

Table 6.34 — *QueryAdjust* command

	Command	Session	UpDn
# of bits	4	2	3
description	1001	00: S0 01: S1 10: S2 11: S3	110: $Q = Q + 1$ 000: No change to Q 011: $Q = Q - 1$

Table 6.35 — Tag reply to a *QueryAdjust* command

	Reply
# of bits	16
description	RN16

An Interrogator shall not encapsulate a *QueryAdjust* in a *SecureComm* or *AuthComm* (see Table 6.28).

Upon receiving a *QueryAdjust* Tags first update *Q*, then pick a random value in the range (0, 2^Q–1), inclusive, and load this value into their slot counter. If a Tag, in response to the *QueryAdjust*, loads its slot counter with zero, then its reply to a *QueryAdjust* shall be shown in Table 6.35 using the *immediate* reply type specified in 6.3.1.6.1; otherwise, the Tag shall remain silent. A Tag shall respond to a *QueryAdjust* only if it received a prior *Query*.

A Tag in any state except **ready** or **killed** shall execute a *QueryAdjust* command if, and only if, (i) the session parameter in the command matches the session parameter in the *Query* that started the round, and (ii) the Tag is not in the middle of a *Kill* or *Access* command sequence (see 6.3.2.12.3.4 or 6.3.2.12.3.6, respectively).

A Tag in the **acknowledged**, **open**, or **secured** state that receives a *QueryAdjust* whose session parameter matches the session parameter in the prior *Query*, and that is not in the middle of a *Kill* or *Access* command sequence (see 6.3.2.12.3.4 or 6.3.2.12.3.6, respectively), shall invert its **inventoried** flag (i.e. *A*→*B* or *B*→*A*, as appropriate) for the current session and transition to **ready**.

6.3.2.12.2.3 *QueryRep* (mandatory)

Interrogators and Tags shall implement the *QueryRep* command shown in Table 6.36. *QueryRep* instructs Tags to decrement their slot counters and, if slot=0 after decrementing, to backscatter an RN16 to the Interrogator.

QueryRep includes the following field:

— Session corroborates the session number for the inventory round (see 6.3.2.8 and 6.3.2.12.2.1). If a Tag receives a *QueryRep* whose session number is different from the session number in the *Query* that initiated the round it shall ignore the command.

An Interrogator shall prepend a *QueryRep* with a frame-sync (see 6.3.1.2.8).

An Interrogator shall not encapsulate a *QueryRep* in a *SecureComm* or *AuthComm* (see Table 6.28).

If a Tag, in response to the *QueryRep*, decrements its slot counter and the decremented slot value is zero, then its reply to a *QueryRep* shall be as shown in Table 6.37 using the *immediate* reply type specified in 6.3.1.6.1; otherwise the Tag shall remain silent. A Tag shall respond to a *QueryRep* only if it received a prior *Query*.

A Tag in any state except **ready** or **killed** shall execute a *QueryRep* command if, and only if, (i) the session parameter in the command matches the session parameter in the *Query* that started the round, and (ii) the Tag is not in the middle of a *Kill* or *Access* command sequence (see 6.3.2.12.3.4 or 6.3.2.12.3.6, respectively).

A Tag in the **acknowledged**, **open**, or **secured** state that receives a *QueryRep* whose session parameter matches the session parameter in the prior *Query*, and that is not in the middle of a *Kill* or *Access* command sequence (see 6.3.2.12.3.4 or 6.3.2.12.3.6, respectively), shall invert its **inventoried** flag (i.e. *A*→*B* or *B*→*A*, as appropriate) for the current session and transition to **ready**.

Table 6.36 — *QueryRep* command

	Command	Session
# of bits	2	2
description	00	00: S0 01: S1 10: S2 11: S3

Table 6.37 — Tag reply to a *QueryRep* command

	Reply
# of bits	16
description	RN16

6.3.2.12.2.4 *ACK* (mandatory)

Interrogators and Tags shall implement the *ACK* command shown in Table 6.38. An Interrogator sends an *ACK* to acknowledge a single Tag. *ACK* echoes the Tag's backscattered RN16.

If an Interrogator issues an *ACK* to a Tag in the **reply** or **acknowledged** state then the echoed RN16 shall be the RN16 that the Tag previously backscattered as it transitioned from the **arbitrate** state to the **reply** state. If an Interrogator issues an *ACK* to a Tag in the **open** or **secured** state then the echoed RN16 shall be the Tag's handle (see 6.3.2.12.2.4).

An Interrogator shall prepend an *ACK* with a frame-sync (see 6.3.1.2.8).

An Interrogator may encapsulate an *ACK* in a *SecureComm* or *AuthComm* (see Table 6.28).

The Tag reply to a successful *ACK* shall be as shown in Table 6.39, using the *immediate* reply type specified in 6.3.1.6.1. As described in 6.3.2.1.2 and shown in Table 6.17, the reply may be truncated or include a concatenated response. A Tag that receives an *ACK* with an incorrect RN16 or an incorrect

handle (as appropriate) shall return to **arbitrate** without responding, unless the Tag is in **ready** or **killed**, in which case it shall ignore the *ACK* and remain in its current state.

If a Tag does not support XPC functionality then the maximum length of its backscattered UII is 496 bits. If a Tag supports XPC functionality then the maximum length of its backscattered UII is reduced by two words to accommodate the optional XPC_W1 and XPC_W2, so is 464 bits (see 6.3.2.1.2.2). In either case a Tag's reply to an *ACK* shall not exceed 528 bits for the PC + UII + PacketCRC, optionally followed by a response field and its associated CRC-16 (see Table 6.17).

Table 6.38 — *ACK* command

	Command	RN
# of bits	2	16
description	01	Echoed RN16 or handle

Table 6.39 — Tag reply to a successful *ACK* command

	Reply
# of bits	21 to 33,328
description	See Table 6.17

6.3.2.12.2.5 *NAK* (mandatory)

Interrogators and Tags shall implement the *NAK* command shown in Table 6.40. A Tag that receives a *NAK* shall return to the **arbitrate** state without changing its **inventoried** flag, unless the Tag is in **ready** or **killed**, in which case it shall ignore the *NAK* and remain in its current state.

An Interrogator shall prepend a *NAK* with a frame-sync (see 6.3.1.2.8).

An Interrogator shall not encapsulate a *NAK* in a *SecureComm* or *AuthComm* (see Table 6.28).

A Tag shall not reply to a *NAK*.

Table 6.40 — *NAK* command

	Command
# of bits	8
description	11000000

6.3.2.12.3 Access commands

The access command set comprises *Req_RN, Read, Write*, *Lock*, *Kill*, *Access*, *BlockWrite*, *BlockErase*, *BlockPermalock*, *Authenticate*, *ReadBuffer*, *SecureComm*, *AuthComm*, *KeyUpdate*, *Untraceable*, *FileOpen, FileList, FilePrivilege, FileSetup*, and *TagPrivilege*.

All access commands include the Tag's handle and a CRC-16. The CRC-16 is calculated over the first command-code bit to the last handle bit. A Tag in the **open** or **secured** state that receives an access command with an incorrect handle but a correct CRC-16 shall behave as specified in Table C.30.

6.3.2.12.3.1 *Req_RN* (mandatory)

Interrogators and Tags shall implement the *Req_RN* command shown in Table 6.41. *Req_RN* instructs a Tag to backscatter a new RN16. Both the Interrogator's command and the Tag's reply depend on the Tag's state:

— **Acknowledged** state: When issuing a *Req_RN* to a Tag in the **acknowledged** state an Interrogator shall include the Tag's last backscattered RN16 as a parameter in the *Req_RN*. The *Req_RN* is

protected by a CRC-16 calculated over the command code and the RN16. If a Tag receives a *Req_RN* with a correct RN16 and a correct CRC-16 then it shall generate and store a new RN16 (denoted handle), backscatter this handle, and transition to the **open** or **secured** state. The choice of ending state depends on the Tag's access password, as follows:

— Access password <> 0: Tag transitions to **open** state.

— Access password = 0: Tag transitions to **secured** state.

A Tag in the **acknowledged** state that receives a *Req_RN* with an incorrect RN16 but a correct CRC-16 shall ignore the *Req_RN* and remain in the **acknowledged** state.

— **Open** or **secured** state: When issuing a *Req_RN* to a Tag in the **open** or **secured** state an Interrogator shall include the Tag's handle as a parameter in the *Req_RN*. If a Tag receives the *Req_RN* with a correct handle and a correct CRC-16 then it shall generate and backscatter a new RN16, remaining in its current state (**open** or **secured**, as appropriate).

If an Interrogator wants to ensure that only one Tag is in the **acknowledged** state then it may issue a *Req_RN*, causing the Tag or Tags to each backscatter a handle and transition to the **open** or **secured** state (as appropriate). The Interrogator may then issue an *ACK* with handle as a parameter in the command. The Tag that receives the *ACK* with a correct handle replies as specified in Table 6.39, whereas those that receive it with an incorrect handle shall return to **arbitrate**. (Note: If a Tag receives an *ACK* with an incorrect handle it returns to **arbitrate**, whereas if it receives an access command with an incorrect handle it behaves as specified in Table C.30).

The first bit of the backscattered RN16 shall be denoted the MSB; the last bit shall be denoted the LSB.

An Interrogator shall prepend a *Req_RN* with a frame-sync (see 6.3.1.2.8).

An Interrogator may encapsulate a *Req_RN* in a *SecureComm* or *AuthComm* (see Table 6.28).

A Tag's reply to a *Req_RN* shall be as shown in Table 6.42, using the *immediate* reply type specified in 6.3.1.6.1. The RN16 or handle are protected by a CRC-16.

Table 6.41 — *Req_RN* command

	Command	RN	CRC
# of bits	8	16	16
description	11000001	Prior RN16 or handle	CRC-16

Table 6.42 — Tag reply to a *Req_RN* command

	RN	CRC
# of bits	16	16
description	handle or new RN16	CRC-16

6.3.2.12.3.2 *Read* (mandatory)

Interrogators and Tags shall implement the *Read* command shown in Table 6.44. A *Read* allows an Interrogator to read part or all of a Tag's Reserved memory, UII memory, TID memory, or the currently open file in User memory. *Read* has the following fields:

— MemBank specifies whether the *Read* accesses Reserved, UII, TID, or User memory. *Read* commands shall apply to a single memory bank. Successive *Reads* may apply to different banks.

— WordPtr specifies the starting word address for the memory read, where words are 16 bits in length. For example, WordPtr=00_h specifies the first 16-bit memory word, WordPtr=01_h specifies the second 16-bit memory word, etc. WordPtr uses EBV formatting (see Annex A).

Table 6.43 — Tag *Read* reply when WordCount=00_h and MemBank=01_2

WordPtr Memory Address	Tag Implements XPC_W1?	Tag Implements XPC_W2?	What the Tag Backscatters
Within the StoredCRC, StoredPC, or the UII specified by bits 10_h–14_h of the StoredPC	Don't care	Don't care	UII memory starting at WordPtr and ending at the UII length specified by StoredPC bits 10_h–14_h.
Within physical UII memory but above the UII specified by bits 10_h–14_h of the StoredPC	No	N/A. See note 1	UII memory starting at WordPtr and ending at the end of physical UII memory, unless the Interrogator has a deasserted Untraceable privilege and the reply would include untraceably hidden memory, in which case error code (see note 2).
	Yes	No	UII memory starting at WordPtr and ending at the end of physical UII memory, unless the Interrogator has a deasserted Untraceable privilege and the reply would include untraceably hidden memory, in which case error code (see note 2). Includes XPC_W1 if WordPtr is less than or equal to 210_h, physical UII memory extends to or above 210_h, and no error code.
	Yes	Yes	UII memory starting at WordPtr and ending at the end of physical UII memory, unless the Interrogator has a deasserted Untraceable privilege and the reply would include untraceably hidden memory, in which case error code (see note 2). Includes XPC_W1 and XPC_W2 if WordPtr is less than or equal to 210_h, physical UII memory extends to or above 210_h, and no error code. Includes XPC_W2 if WordPtr is equal to 220_h and physical UII memory extends to or above 220_h.
210_h. Above physical UII memory	No	N/A. See note 1	Error code.
	Yes	No	XPC_W1.
	Yes	Yes	XPC_W1 and XPC_W2.
220_h. Above physical UII memory	No	N/A. See note 1	Error code.
	Yes	No	Error code.
	Yes	Yes	XPC_W2.
Not 210_h or 220_h. Above physical UII memory.	Don't care	Don't care	Error code.

Note 1 If a Tag does not implement an XPC_W1 then it does not implement an XPC_W2. See 6.3.2.1.2.5.

Note 2 Untraceably hidden memory is not readable except by an Interrogator with an asserted Untraceable privilege. See 6.3.2.12.3.16.

— WordCount specifies the number of 16-bit words to read. If WordCount=00_h then a Tag shall backscatter the contents of the chosen memory bank or file starting at WordPtr and ending at the end of the memory bank or file, however:

  — if MemBank=01_2 then a Tag shall backscatter the memory contents specified in Table 6.43.

  — if MemBank=10_2, and part of TID memory is untraceably hidden (see 6.3.2.12.3.16), and the Interrogator has a deasserted Untraceable privilege, and the memory address specified by WordPtr is in the traceable part of TID memory, then a Tag may either (i) backscatter the

traceable part of TID memory starting at WordPtr, or (ii) treat the command's parameters as unsupported (see Table C.30), depending on the Tag manufacturer's implementation.

An Interrogator shall prepend a *Read* with a frame-sync (see 6.3.1.2.8).

An unauthenticated Interrogator may, and an authenticated Interrogator shall, encapsulate a *Read* command in a *SecureComm* or *AuthComm* (see Table 6.28).

A Tag shall reply to a *Read* using the *immediate* reply type specified in 6.3.1.6.1. If all memory words specified in a *Read* exist, none are read-locked, all are traceable or the Interrogator has an asserted Untraceable privilege, and for User memory the Interrogator has read privileges to the currently open file (see 6.3.2.11.3), then a Tag's reply to a *Read* shall be as shown in Table 6.45 comprising a header (a 0-bit), the requested memory words, and the Tag's handle. The reply includes a CRC-16 calculated over the 0-bit, memory words, and handle. Otherwise the Tag shall not execute the *Read* and instead treat the command's parameters as unsupported (see Table C.30).

Table 6.44 — *Read* command

	Command	MemBank	WordPtr	WordCount	RN	CRC
# of bits	8	2	EBV	8	16	16
description	11000010	00: Reserved 01: UII 10: TID 11: User	Starting address pointer	Number of words to read	handle	CRC-16

Table 6.45 — Tag reply to a successful *Read* command

	Header	Memory Words	RN	CRC
# of bits	1	Variable	16	16
description	0	Data	handle	CRC-16

6.3.2.12.3.3 *Write* (mandatory)

Interrogators and Tags shall implement the *Write* command shown in Table 6.46. *Write* allows an Interrogator to write a word in a Tag's Reserved memory, UII memory, TID memory, or the currently open file in User memory. *Write* has the following fields:

— MemBank specifies whether the *Write* occurs in Reserved, UII, TID, or User memory. *Write* commands shall apply to a single memory bank. Successive *Writes* may apply to different banks.

— WordPtr specifies the word address for the memory write, where words are 16 bits in length. For example, WordPtr=00_h specifies the first 16-bit memory word, WordPtr=01_h specifies the second 16-bit memory word, etc. WordPtr uses EBV formatting (see Annex A).

— Data contains a 16-bit word to be written. Before each and every *Write* the Interrogator shall first issue a *Req_RN* command; the Tag replies by backscattering a new RN16. The Interrogator shall cover code the data by EXORing it with this new RN16 prior to transmission.

A Tag shall only execute a *Write* in the **open** or **secured** state. If a Tag in the **open** or **secured** state receives a *Write* before which the immediately preceding command was not a *Req_RN* then it shall not execute the *Write* and instead treat the command as invalid (see Table C.30).

If an Interrogator attempts to write to the kill or access password, UII or TID memory banks, or File_0 and these memory locations are permalocked; or to the kill or access password, UII or TID memory banks, or File_0 and these memory locations are locked unwriteable and the Tag is in the **open** state; or to a permalocked block in File_N, N≥0 of User memory; or to memory that is untraceably hidden and the Interrogator has a deasserted Untraceable privilege; or to a file for which the Interrogator does not

have sufficient privileges; then the Tag shall not execute the *Write* and instead treat the command's parameters as unsupported (see Table C.30).

An Interrogator shall prepend a *Write* with a frame-sync (see 6.3.1.2.8).

An Interrogator shall not encapsulate a *Write* in a *SecureComm* or *AuthComm* (see Table 6.28).

Upon receiving an executable *Write* a Tag shall write the commanded data into memory.

Table 6.46 — *Write* command

	Com-mand	MemBank	WordPtr	Data	RN	CRC
# of bits	8	2	EBV	16	16	16
description	11000011	00: Reserved 01: UII 10: TID 11: User	Address pointer	RN16 ⊗ word to be written	handle	CRC-16

A Tag shall reply to a *Write* using the *delayed* reply specified in 6.3.1.6.2.

6.3.2.12.3.4 *Kill* (mandatory)

Interrogators and Tags shall implement the *Kill* command shown in Table 6.47. *Kill* allows an Interrogator to permanently disable a Tag. To kill a Tag, an Interrogator shall follow the kill procedure shown in Figure 6.24. A Tag shall implement the password-based kill sequence shown in the left-side branch of the kill procedure in Figure 6.24. A Tag that implements Interrogator or mutual authentication and *SecureComm* or *AuthComm* may also implement the authenticated-kill sequence shown in the right-side branch of the kill procedure in Figure 6.24. *Kill* has the following fields:

— Password specifies half of the kill password EXORed with an RN16.

A *Kill* contains 3 RFU bits. An Interrogator shall set these bits to 000_2. A Tag shall ignore these bits. Future protocols may use these bits to expand the functionality of the *Kill* command.

An Interrogator shall prepend an unencapsulated *Kill* command with a frame-sync (see 6.3.1.2.8).

An Interrogator may encapsulate a *Kill* in a *SecureComm* or *AuthComm* (see Table 6.28).

Password-based kill (mandatory)

A Tag may execute a password-based kill from the **open** or **secured** state. A Tag is not required to authenticate an Interrogator for a password-based kill. To perform the kill, an Interrogator issues two successive *Kill* commands, the first containing the 16 MSBs of the Tag's kill password EXORed with an RN16, and the second containing the 16 LSBs of the Tag's kill password EXORed with a different RN16. Each EXOR operation shall be performed MSB first (i.e. the MSB of each half-password shall be EXORed with the MSB of its respective RN16). Just prior to issuing each *Kill* command the Interrogator first issues a *Req_RN* to obtain a new RN16.

A Tag shall be capable of successively accepting two 16-bit subportions of the 32-bit kill password. An Interrogator shall not intersperse commands other than a *Req_RN* between the two successive *Kill* commands. If a Tag, after receiving a first *Kill*, receives any valid command other than *Req_RN* before the second *Kill* then it shall not execute the command and instead treat is as improper (see Table C.30), unless the intervening command is a *Query*, in which case the Tag shall execute the *Query* and invert its **inventoried** flag if the session parameter in the *Query* matches that in the prior session.

A Tag with a zero-valued kill password shall disallow itself from being killed by a password-based kill operation. A Tag with a zero-valued kill password shall respond to a password-based kill by not executing the kill operation and backscattering an error code, remaining in its current state. See Figure 6.24.

A Tag shall reply to a first *Kill* using the *immediate* reply specified in 6.3.1.6.1. The Tag's first reply shall be as shown in Table 6.48. The reply shall use the TRext value specified in the *Query* command that initiated the round.

A Tag shall reply to the second *Kill* using the *delayed* reply specified in 6.3.1.6.2. If the kill succeeds then the Tag, after sending the final reply shown in Table 6.13, shall render itself silent and shall not respond to an Interrogator thereafter. If the kill does not succeed then the Interrogator may issue a *Req_RN* containing the Tag's handle to verify that the Tag is in the Interrogator's field, and may again attempt the multi-step kill procedure in Figure 6.24.

If a Tag observes a properly formatted password-based *Kill*-command sequence but the kill fails (as will happen if the Interrogator sends an incorrect kill password) then the Tag shall return to **arbitrate** and may enforce a security timeout as specified in 6.3.2.5. If a Tag that supports security timeouts for a password-based *Kill*-command sequence receives such a sequence during a timeout then it shall behave as though it is not killable, backscatter an error code (see Annex I), and remain in its current state.

Authenticated kill (optional)

A Tag may execute an authenticated kill from the **secured** state. A Tag shall authenticate an Interrogator via an Interrogator or mutual authentication prior to executing an authenticated kill. To perform an authenticated kill an Interrogator issues a single *Kill* command encapsulated in a *SecureComm* or *AuthComm*. The Interrogator may use any 16-bit value in the password field of the *Kill* command because a Tag shall ignore the kill password for an authenticated kill. An Interrogator is not required to issue a *Req_RN* prior to sending an encapsulated *Kill*.

A Tag shall only execute an authenticated kill if the Interrogator possesses an asserted AuthKill privilege (see Table 6.23) and the Tag is in the **secured** state. A Tag shall reply to an authenticated kill using an in-process reply (as required by a *SecureComm* or *AuthComm*), but with SenRep=1 regardless of the SenRep value actually specified in the *SecureComm* or *AuthComm*. If the kill succeeds then the Tag, after sending the final reply shown in Table 6.13, shall transition to the **killed** state and not respond to an Interrogator thereafter. If the kill fails then the Tag shall remain in its current state and backscatter an error code (see Annex I), unless the Tag is in the **open** state, the Interrogator is not authenticated, or the Interrogator does not have an asserted AuthKill privilege (see Table 6.23), in which case the Tag shall return to **arbitrate** and may enforce a security timeout as specified in 6.3.2.5. If a Tag that supports security timeouts for an authenticated *Kill* command receives an authenticated *Kill* command during a timeout then it shall behave as though it is not killable, backscatter an error code (see Annex I), and remain in its current state.

Table 6.47 — *Kill* command

	Command	Password	RFU	RN	CRC
# of bits	8	16	3	16	16
description	11000100	Password-based kill: (½ kill password) ⊗ RN16 Authenticated kill: any 16-bit value	000_2	handle	CRC-16

Table 6.48 — Tag reply to the first *Kill* command

	RN	CRC
# of bits	16	16
description	handle	CRC-16

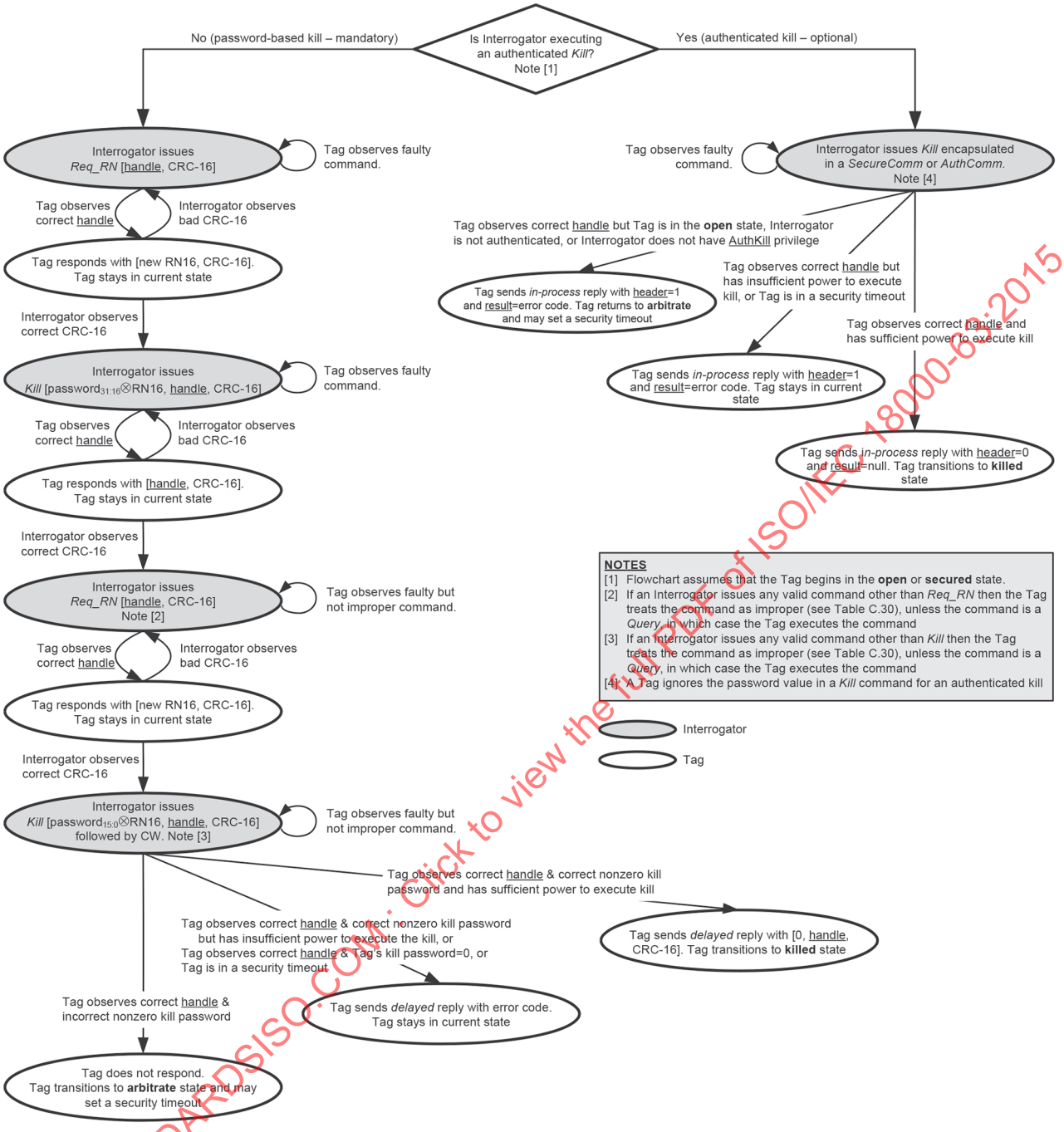


Figure 6.24 — *Kill* procedure

6.3.2.12.3.5 *Lock* (mandatory)

Interrogators and Tags shall implement the *Lock* command shown in Table 6.49 and Figure 6.25. *Lock* allows an Interrogator to:

— Lock the kill and/or access passwords, thereby preventing or allowing subsequent reads and/or writes of those passwords,

— Lock the UII and TID memory banks, thereby preventing or allowing subsequent writes to those banks,

— Lock File_0 of User memory, thereby preventing or allowing subsequent writes to File_0, and

— Make the lock status for the passwords, UII memory, TID memory, and/or File_0 permanent.

Lock contains a 20-bit payload defined as follows:

— The first 10 payload bits are Mask bits. A Tag shall interpret these bit values as follows:

  — Mask=0: Ignore the associated Action field and retain the current lock setting.

  — Mask=1: Implement the associated Action field and overwrite the current lock setting.

— The last 10 payload bits are Action bits. A Tag shall interpret these bit values as follows:

  — Action=0: Deassert lock for the associated memory location.

  — Action=1: Assert lock or permalock for the associated memory location.

The functionality of the various Action fields is described in Table 6.50.

The payload of a *Lock* command shall always be 20 bits in length.

If an Interrogator issues a *Lock* whose Mask and Action fields attempt to change the lock status of a nonexistent memory bank, nonexistent File_0, or nonexistent password then a Tag shall not execute the *Lock* and instead treat the command's parameters as unsupported (see Table C.30).

Lock differs from *BlockPermalock* in that *Lock* reversibly or permanently locks the kill and/or access password, the UII memory bank, the TID memory bank, and/or File_0 of User memory in a writeable or unwriteable state, whereas *BlockPermalock* permanently locks individual blocks of File_N, N≥0 of User memory in an unwriteable state. Table 6.55 specifies how a Tag reacts to a *Lock* targeting File_0 that follows a prior *BlockPermalock* (with Read/Lock=1), or vice versa.

Permalock bits, once asserted, cannot be deasserted. If a Tag receives a *Lock* whose payload attempts to deassert a previously asserted permalock bit then the Tag shall not execute the *Lock* and instead treat the command's parameters as unsupported (see Table C.30). If a Tag receives a *Lock* whose payload attempts to reassert a previously asserted permalock bit then the Tag shall ignore this particular Action field and implement the remainder of the *Lock* payload.

An *Untraceable* command may change the values of the **L** and **U** bits in UII memory regardless of the lock or permalock status of the UII memory bank. See 6.3.2.12.3.16.

A Tag manufacturer may choose where a Tag stores its lock bits and may choose a readable portion of memory (if desired). Regardless of the location, a field-deployed Tag shall not permit an Interrogator to change its lock bits except by means of a *Lock* command.

A Tag shall implement memory locking and the *Lock* command. However, a Tag need not support all the Action fields in Figure 6.25, depending on whether a Tag implements the memory location associated with the Action field and that memory location is lockable and/or unlockable. If a Tag receives a *Lock* it cannot execute because one or more memory locations do not exist, or one or more of the Action fields attempt to change a permalocked value, or one or more of the memory locations are either not lockable or not unlockable, then the Tag shall not execute the *Lock* and instead treat the command's parameters as unsupported (see Table C.30). The only exception to this general rule is for a Tag that (a) does not support File_N, N>0 and (b) whose only lock functionality is to permanently lock **all** memory (i.e. all memory banks and all passwords) at once; such a Tag shall execute a *Lock* whose payload is $FFFFF_h$, and shall backscatter an error code for any payload other than $FFFFF_h$.

A Tag in the **secured** state shall permit an Interrogator to write or erase memory locations with (pwd-write=1 AND permalock=0) or (pwd-read/write=1 AND permalock=0) without first issuing a *Lock* to change these fields.

An Interrogator shall prepend a *Lock* with a frame-sync (see 6.3.1.2.8).

An unauthenticated Interrogator may, and an authenticated Interrogator shall, encapsulate a *Lock* command in a *SecureComm* or *AuthComm* (see Table 6.28).

Upon receiving an executable *Lock* a Tag shall perform the commanded lock operation. A Tag shall reply to a *Lock* using the *delayed* reply specified in 6.3.1.6.2.

Table 6.49 — *Lock* command

	Command	Payload	RN	CRC
# of bits	8	20	16	16
description	11000101	Mask and Action Fields	handle	CRC-16

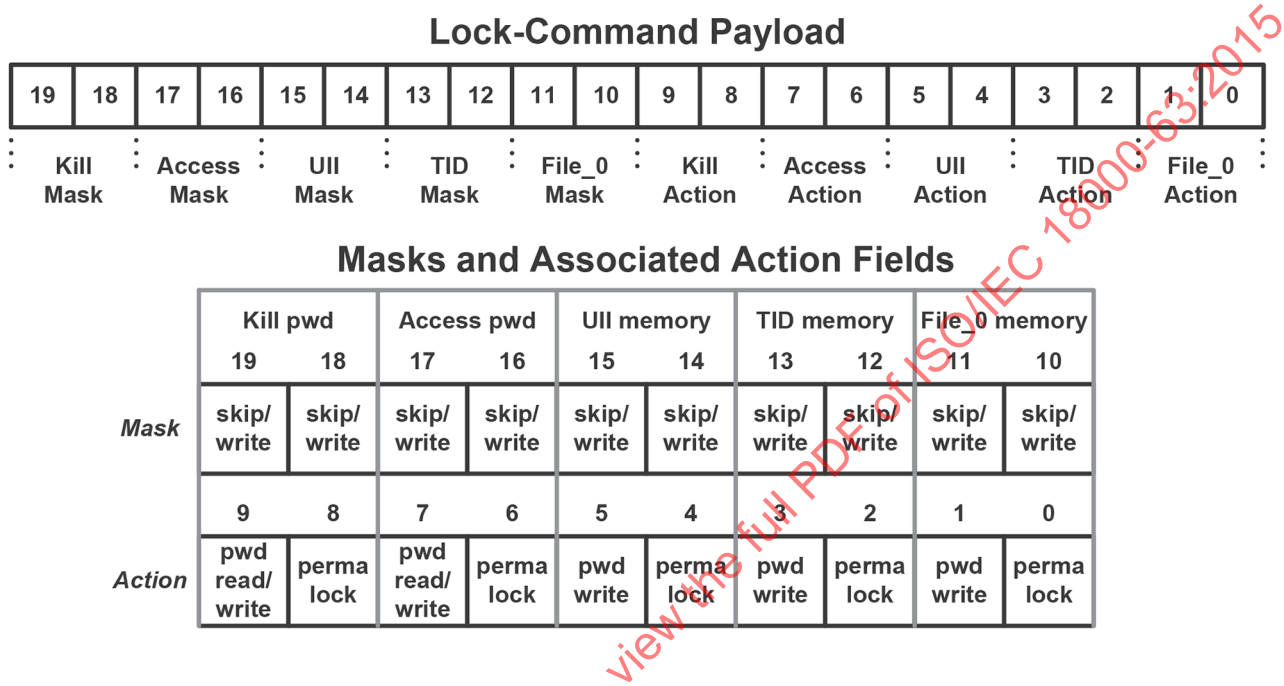


Figure 6.25 — *Lock* payload and usage

Table 6.50 — *Lock* Action-field functionality

pwd-write	permalock	Description
0	0	Associated memory bank/file is writeable from either the **open** or **secured** states.
0	1	Associated memory bank/file is permanently writeable from either the **open** or **secured** states and may never be locked.
1	0	Associated memory bank/file is writeable from the **secured** state but not from the **open** state.
1	1	Associated memory bank/file is not writeable from any state.
pwd-read/ write	**permalock**	**Description**
0	0	Associated password location is readable and writeable from either the **open** or **secured** states.
0	1	Associated password location is permanently readable and writeable from either the **open** or **secured** states and may never be locked.
1	0	Associated password location is readable and writeable from the **secured** state but not from the **open** state.
1	1	Associated password location is not readable or writeable from any state.

6.3.2.12.3.6 *Access* (optional)

Interrogators and Tags may implement an *Access* command; if they do, they shall implement it as shown in Table 6.51. *Access* allows an Interrogator to transition a Tag from the **open** to the **secured** state or, if the Tag is already in the **secured** state, to remain in **secured**. *Access* has the following fields:

— Password specifies half of the access password EXORed with an RN16.

To access a Tag, an Interrogator shall follow the multi-step procedure outlined in Figure 6.26. Briefly, an Interrogator issues two *Access* commands, the first containing the 16 MSBs of the Tag's access password EXORed with an RN16, and the second containing the 16 LSBs of the Tag's access password EXORed with a different RN16. Each EXOR operation shall be performed MSB first (i.e. the MSB of each half-password shall be EXORed with the MSB of its respective RN16). Just prior to issuing each *Access* the Interrogator first issues a *Req_RN* to obtain a new RN16.

A Tag shall be capable of successively accepting two 16-bit subportions of the 32-bit access password. An Interrogator shall not intersperse commands other than a *Req_RN* between the two successive *Access* commands. If a Tag, after receiving a first *Access*, receives any valid command other than *Req_RN* before the second *Access* then it shall not execute the command and instead treat it as improper (see Table C.30), unless the intervening command is a *Query*, in which case the Tag shall execute the *Query* and invert its **inventoried** flag if the session parameter in the *Query* matches that in the prior session.

An Interrogator shall prepend an *Access* with a frame-sync (see 6.3.1.2.8).

An Interrogator shall not encapsulate an *Access* in a *SecureComm* or *AuthComm* (see Table 6.28).

A Tag shall reply to an *Access* using the *immediate* reply specified in 6.3.1.6.1. The reply shall be as shown in Table 6.52. If the *Access* is the first in the sequence then the Tag backscatters its handle to acknowledge that it received the command. If the *Access* is the second in the sequence and the received 32-bit access password is correct then the Tag backscatters its handle to acknowledge that it has executed the command successfully and has transitioned to the **secured** state; otherwise the Tag does not reply and returns to **arbitrate**. The Tag reply includes a CRC-16 calculated over the handle.

Table 6.51 — *Access* command

	Command	Password	RN	CRC
# of bits	8	16	16	16
description	11000110	(½ access password) ⊗ RN16	handle	CRC-16

Table 6.52 — Tag reply to an *Access* command

	RN	CRC
# of bits	16	16
description	handle	CRC-16

If a Tag observes a properly formatted *Access* sequence but the Interrogator sends an incorrect access password then the Tag shall return to **arbitrate** and may enforce a security timeout as specified in 6.3.2.5. If a Tag that supports security timeouts for an *Access* command sequence receives such a sequence during a timeout then it shall behave as though Tag access was disallowed, backscatter an error code (see Annex I), and remain in its current state.

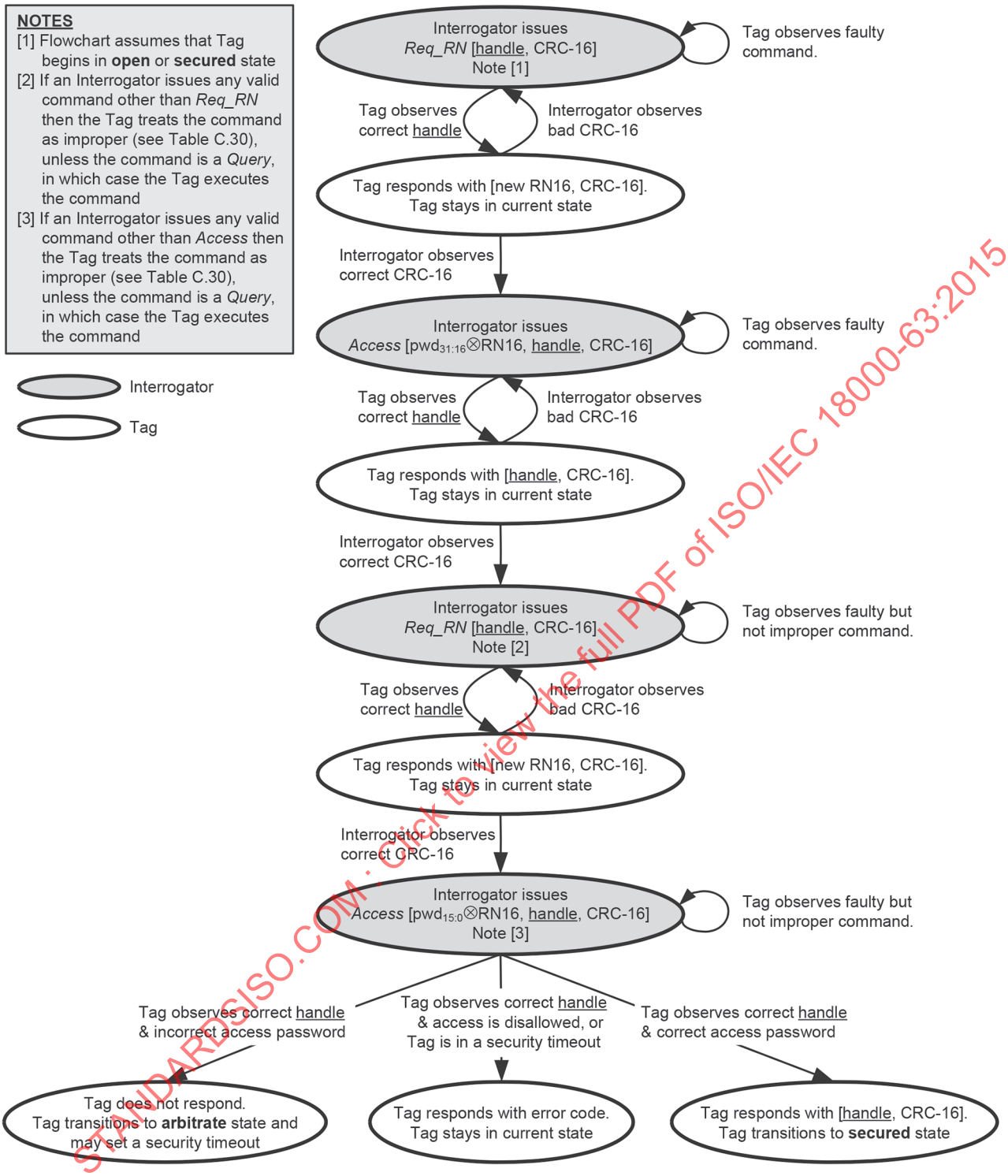


Figure 6.26 — *Access* procedure

6.3.2.12.3.7 *BlockWrite* (optional)

Interrogators and Tags may implement a *BlockWrite* command; if they do, they shall implement it as shown in Table 6.53. *BlockWrite* allows an Interrogator to write multiple words in a Tag's Reserved

memory, UII memory, TID memory, or the currently open file in User memory. *BlockWrite* has the following fields:

— MemBank specifies whether the *BlockWrite* occurs in Reserved, UII, TID, or User memory. *BlockWrite* commands shall apply to a single memory bank. Successive *BlockWrites* may apply to different banks.

— WordPtr specifies the starting word address for the memory write, where words are 16 bits in length. For example, WordPtr=00_h specifies the first 16-bit memory word, WordPtr=01_h specifies the second 16-bit memory word, etc. WordPtr uses EBV formatting (see Annex A).

— WordCount specifies the number of 16-bit words to be written. If WordCount=00_h then a Tag shall treat the *BlockWrite* as invalid. If WordCount=01_h then a Tag shall write a single data word.

— Data contains the 16-bit words to be written, and shall be 16×WordCount bits in length. Unlike a *Write*, the data in a *BlockWrite* are not cover-coded, and an Interrogator need not issue a *Req_RN* before issuing a *BlockWrite*.

A Tag shall only execute a *BlockWrite* in the **open** or **secured** state.

If an Interrogator attempts to write to the kill or access password, UII or TID memory banks, or File_0 and these memory locations are permalocked; or to the kill or access password, UII or TID memory banks, or File_0 and these memory locations are locked unwriteable and the Tag is in the **open** state; or to memory that is untraceably hidden and the Interrogator has a deasserted Untraceable privilege; or to a file for which the Interrogator does not have sufficient privileges; or if WordPtr and WordCount include one or more permalocked blocks in File_N, N≥0 of User memory; then the Tag shall not execute the *BlockWrite* and instead treat the command's parameters as unsupported (see Table C.30).

An Interrogator shall prepend a *BlockWrite* with a frame-sync (see 6.3.1.2.8).

An unauthenticated Interrogator may, and an authenticated Interrogator shall, encapsulate a *BlockWrite* in a *SecureComm* or *AuthComm* (see Table 6.28).

Upon receiving an executable *BlockWrite* a Tag shall write the commanded data into memory.

Table 6.53 — *BlockWrite* command

	Command	MemBank	WordPtr	WordCount	Data	RN	CRC
# of bits	8	2	EBV	8	Variable	16	16
description	11000111	00: Reserved 01: UII 10: TID 11: User	Starting address pointer	Number of words to write	Data to be written	handle	CRC-16

A Tag shall reply to a *BlockWrite* using the *delayed* reply specified in 6.3.1.6.2.

6.3.2.12.3.8 *BlockErase* (optional)

Interrogators and Tags may implement a *BlockErase* command; if they do, they shall implement it as shown in Table 6.54. *BlockErase* allows an Interrogator to erase multiple words in a Tag's Reserved memory, UII memory, TID memory, or the currently open file in User memory. *BlockErase* has the following fields:

— MemBank specifies whether the *BlockErase* occurs in Reserved, UII, TID, or User memory. *BlockErase* commands shall apply to a single memory bank. Successive *BlockErases* may apply to different banks.

— WordPtr specifies the starting word address for the memory erase, where words are 16 bits in length. For example, WordPtr=00_h specifies the first 16-bit memory word, WordPtr=01_h specifies the second 16-bit memory word, etc. WordPtr uses EBV formatting (see Annex A).

— WordCount specifies the number of 16-bit words to be erased. If WordCount=00_h then a Tag shall treat the *BlockErase* as invalid. If WordCount=01_h then a Tag shall erase a single data word.

A Tag shall only execute a *BlockErase* in the **open** or **secured** state.

If an Interrogator attempts to erase the kill or access password, UII or TID memory banks, or File_0 and these memory locations are permalocked; or the kill or access password, UII or TID memory banks, or File_0 and these memory locations are locked unwriteable and the Tag is in the **open** state; or to memory that is untraceably hidden and the Interrogator has a deasserted Untraceable privilege; or a file for which the Interrogator does not have sufficient privileges; or if WordPtr and WordCount include one or more permalocked blocks in File_N, N≥0 of User memory; then the Tag shall not execute the *BlockErase* and instead treat the command's parameters as unsupported (see Table C.30).

An Interrogator shall prepend a *BlockErase* with a frame-sync (see 6.3.1.2.8).

An unauthenticated Interrogator may, and an authenticated Interrogator shall, encapsulate a *BlockErase* in a *SecureComm* or *AuthComm* (see Table 6.28).

Table 6.54 — *BlockErase* command

	Command	MemBank	WordPtr	WordCount	RN	CRC
# of bits	8	2	EBV	8	16	16
description	11001000	00: Reserved 01: UII 10: TID 11: User	Starting address pointer	Number of words to erase	handle	

Upon receiving an executable *BlockErase* command a Tag shall erase the commanded memory words.

A Tag shall reply to a *BlockErase* using the *delayed* reply specified in 6.3.1.6.2.

6.3.2.12.3.9 BlockPermalock (optional)

Interrogators and Tags may implement a *BlockPermalock* command; if they do, they shall implement it as shown in Table 6.56. *BlockPermalock* allows an Interrogator to:

— Permalock one or more memory blocks in the currently open file of a Tag's User memory, or

— Read the permalock status of the memory blocks in the currently open file of a Tag's User memory.

A *BlockPermalock* may permalock between zero and 4080 memory blocks. The block size, which is predefined by the Tag manufacturer, is fixed at between one and 1024 words, is the same for all files, and is the same for block permalocking and file allocation. The memory blocks specified by a *BlockPermalock* need not be contiguous.

A Tag shall only execute a *BlockPermalock* in the **secured** state.

Table 6.55 — Precedence for *Lock* and *BlockPermalock* targeting File_0

<table>
<tr><th colspan="3">First Command</th><th colspan="3">Second Command</th><th>Tag Action and Response to 2nd Command</th></tr>
<tr><td rowspan="5">Lock</td><td>pwd-write</td><td>permalock</td><td colspan="3" rowspan="5">BlockPermalock
(Read/Lock=1)</td><td></td></tr>
<tr><td>0</td><td>0</td><td>Permalock the blocks indicated by Mask; respond as described in this section 6.3.2.12.3.9</td></tr>
<tr><td>0</td><td>1</td><td>Do not execute the BlockPermalock; respond with an error code (Table C.30, unsupported parameters)</td></tr>
<tr><td>1</td><td>0</td><td>Permalock the blocks indicated by Mask; respond as described in this section 6.3.2.12.3.9</td></tr>
<tr><td>1</td><td>1</td><td>Permalock the blocks indicated by Mask; respond as described in this section 6.3.2.12.3.9</td></tr>
<tr><td colspan="3" rowspan="5">BlockPermalock
(Read/Lock=1)</td><td rowspan="5">Lock</td><td>pwd-write</td><td>permalock</td><td></td></tr>
<tr><td>0</td><td>0</td><td>Implement the Lock, but do not un-permalock any blocks that were previously permalocked; respond as described in 6.3.2.12.3.5</td></tr>
<tr><td>0</td><td>1</td><td>Implement the Lock, but do not un-permalock any blocks that were previously permalocked; respond as described in 6.3.2.12.3.5</td></tr>
<tr><td>1</td><td>0</td><td>Implement the Lock, but do not un-permalock any blocks that were previously permalocked; respond as described in 6.3.2.12.3.5</td></tr>
<tr><td>1</td><td>1</td><td>Implement the Lock; respond as described in 6.3.2.12.3.5</td></tr>
</table>

A *BlockPermalock* differs from a *Lock* in that *BlockPermalock* permanently locks individual blocks of File_N, N≥0 of User memory in an unwriteable state whereas *Lock* reversibly or permanently locks the kill and/or access password, the UII memory bank, the TID memory bank, and/or File_0 of User memory in a writeable or unwriteable state. Table 6.55 specifies how a Tag shall behave upon receiving a *BlockPermalock* targeting File_0 that follows a prior *Lock*, or vice versa (assuming Read/Lock=1).

A *BlockPermalock* has the following fields:

— MemBank specifies whether the *BlockPermalock* applies to UII, TID, or User memory. *BlockPermalock* commands shall apply to a single memory bank. Successive *BlockPermalocks* may apply to different memory banks. A Tag shall only execute a *BlockPermalock* command if MemBank=11 (User memory); if a Tag receives a *BlockPermalock* with MemBank<>11 then it shall not execute the *BlockPermalock* and instead treat the command's parameters as unsupported (see Table C.30). Future protocols may use these other MemBank values to expand the functionality of the *BlockPermalock* command.

— Read/Lock specifies whether a Tag backscatters the permalock status of, or permalocks, one or more blocks within the memory bank specified by MemBank. A Tag shall interpret the Read/Lock bit as follows:

  — Read/Lock=0: A Tag shall backscatter the permalock status of blocks in the specified memory bank, starting from the memory block located at BlockPtr and ending at the memory block located at BlockPtr+(16×BlockRange)–1. A Tag shall backscatter a "0" if the memory block corresponding to that bit is not permalocked and a "1" if the block is permalocked. An Interrogator omits Mask from the *BlockPermalock* when Read/Lock=0.

  - Read/Lock=1: A Tag shall permalock those blocks in the specified memory bank that are specified by Mask, starting at BlockPtr and ending at BlockPtr+(16×BlockRange)–1.

- BlockPtr specifies the starting address for Mask, in units of 16 blocks. For example, BlockPtr=00_h indicates block 0, BlockPtr=01_h indicates block 16, BlockPtr=02_h indicates block 32. BlockPtr uses EBV formatting (see Annex A).

- BlockRange specifies the range of Mask, starting at BlockPtr and ending (16×BlockRange)–1 blocks later. If BlockRange=00_h then a Tag shall not execute the *BlockPermalock* and instead treat the command's parameters as unsupported (see Table C.30).

- Mask specifies which memory blocks a Tag permalocks. Mask depends on the Read/Lock bit as follows:

  - Read/Lock=0: The Interrogator shall omit Mask from the *BlockPermalock*.

  - Read/Lock=1: The Interrogator shall include a Mask of length 16×BlockRange bits in the *BlockPermalock*. The Mask bits shall be ordered from lower-order block to higher (i.e. if BlockPtr=00_h then the leading Mask bit refers to block 0). The Tag shall interpret each bit of Mask as follows:

- Mask bit=0: Retain the current permalock setting for the corresponding memory block.

- Mask bit=1: Permalock the corresponding memory block. If a block is already permalocked then a Tag shall retain the current permalock setting. A memory block, once permalocked, cannot be un-permalocked.

The following example illustrates the usage of Read/Lock, BlockPtr, BlockRange, and Mask: If Read/Lock=1, BlockPtr=01_h, and BlockRange=01_h then the Tag operates on sixteen blocks starting at block 16 and ending at block 31, permalocking those blocks whose corresponding bits are asserted in Mask.

A *BlockPermalock* contains 8 RFU bits. An Interrogator shall set these bits to 00_h. A Tag in the **secured** state that receives a *BlockPermalock* with nonzero RFU bits shall not execute the *BlockPermalock* and instead treat the command's parameters as unsupported (see Table C.30). Future protocols may use these RFU bits to expand the *BlockPermalock* command's functionality.

If a Tag receives a *BlockPermalock* that it cannot execute because User memory does not exist, or User memory is untraceably hidden and the Interrogator has a deasserted Untraceable privilege, or in which one of the asserted Mask bits references a non-existent memory block, or because the Interrogator has insufficient file privileges (see 6.3.2.11.3) then the Tag shall not execute the *BlockPermalock* and instead treat the command's parameters as unsupported (see Table C.30). A Tag shall treat as invalid a *BlockPermalock* in which Read/Lock=0 but Mask is not omitted, or a *BlockPermalock* in which Read/Lock=1 but Mask has a length not equal to 16×BlockRange bits (see Table C.30).

Certain Tags, depending on the Tag manufacturer's implementation, may be unable to execute a *BlockPermalock* with certain BlockPtr and BlockRange values, in which case the Tag shall not execute the *BlockPermalock* and instead treat the command's parameters as unsupported (see Table C.30). Because a Tag contains information in its TID memory that an Interrogator can use to identify the optional features that the Tag supports (see 6.3.2.1.3), this protocol recommends that an Interrogator read a Tag's TID memory prior to issuing a *BlockPermalock*.

If an Interrogator issues a *BlockPermalock* in which BlockPtr and BlockRange specify one or more nonexistent blocks, but Mask only asserts permalocking on existent blocks, then the Tag shall execute the *BlockPermalock*.

An Interrogator shall prepend a *BlockPermalock* with a frame-sync (see 6.3.1.2.8).

An unauthenticated Interrogator may, and an authenticated Interrogator shall, encapsulate a *BlockPermalock* in a *SecureComm* or *AuthComm* (see Table 6.28).

Upon receiving an executable *BlockPermalock* a Tag shall perform the requested operation, unless the Tag does not support block permalocking in which case it shall treat the command as invalid (see Table C.30).

If Read/Lock=0 then a Tag shall reply to a *BlockPermalock* using the *immediate* reply type specified in 6.3.1.6.1. If the Tag is able to execute the *BlockPermalock* then its reply shall be as shown in Table 6.57 comprising a header (a 0-bit), the requested permalock bits, and the Tag's handle. The reply includes a CRC-16 calculated over the 0-bit, permalock bits, and handle. If the Tag is unable to execute the *BlockPermalock* then it shall backscatter an error code (see Table C.30, unsupported parameters) rather than the reply shown in Table 6.57. The Tag's reply when Read/Lock=0 shall use the preamble specified by the TRext value in the *Query* that initiated the inventory round.

If Read/Lock=1 then a Tag shall reply to a *BlockPermalock* using the *delayed* reply specified in 6.3.1.6.2.

Table 6.56 — *BlockPermalock* command

	Command	RFU	Read/Lock	Mem-Bank	BlockPtr	Block-Range	Mask	RN	CRC
# of bits	8	8	1	2	EBV	8	Variable	16	16
description	11001001	00_h	0: Read 1: Permalock	00: RFU 01: UII 10: TID 11: User	Mask starting address, specified in units of 16 blocks	Mask range, specified in units of 16 blocks	0: Retain current permalock setting 1: Assert perma-lock	handle	CRC-16

Table 6.57 — Tag reply to a successful *BlockPermalock* command with Read/Lock=0

	Header	Data	RN	CRC
# of bits	1	Variable	16	16
description	0	Permalock bits	handle	CRC-16

6.3.2.12.3.10 Authenticate (optional)

Interrogators and Tags may implement the *Authenticate* command; if they do, they shall implement it as shown in Table 6.58. *Authenticate* allows an Interrogator to perform Tag, Interrogator, or mutual authentication. The generic nature of the *Authenticate* command allows it to support a variety of cryptographic suites. The CSI specified in an *Authenticate* selects one cryptographic suite from among those supported by the Tag. The number of *Authenticate* commands required to implement an authentication depends on the authentication type and on the chosen cryptographic suite. A Tag only executes an *Authenticate* in the **open** or **secured** state. *Authenticate* has the following fields:

— SenRep specifies whether a Tag backscatters its response or stores the response in its ResponseBuffer.

— IncRepLen specifies whether a Tag omits or includes length in its reply. If IncRepLen=0 then a Tag omits length from its reply; if IncRepLen=1 then the Tag includes length in its reply.

— CSI selects the cryptographic suite that Tag and Interrogator use for the authentication as well as for all subsequent communications (until the Interrogator initiates another authentication with a different CSI or the Tag leaves the **open** or **secured** state).

— Length is the message length in bits.

— Message includes parameters for the authentication.

An *Authenticate* contains 2 RFU bits. An Interrogator shall set these bits to 00_2. A Tag in the **open** or **secured** states that receives an *Authenticate* with nonzero RFU bits shall not execute the *Authenticate* and instead treat the command's parameters as unsupported (see Table C.30). Future protocols may use these RFU bits to expand the *Authenticate* command's functionality.

An Interrogator shall prepend an *Authenticate* with a frame-sync (see 6.3.1.2.8).

An Interrogator shall not encapsulate an *Authenticate* in a *SecureComm* or *AuthComm* (see Table 6.28).

If a Tag supports the *Authenticate* command then it shall implement the security (**S**) indicator (see 6.3.2.1.3).

The cryptographic suite specifies message formatting, the number of steps in an authentication, whether an authentication implements wait states, the behavior if Tag or Interrogator cannot complete a computation, and the behavior in the event of an incorrect cryptographic response. It specifies the formatting of the Tag's response for both a successful and an unsuccessful authentication. It may include parameters, such as a key, that affect pre- and post-authenticated communications. It may contain information about how Tag and Interrogator derive session keys for subsequent communications. See Annex M for the parameters specified by a cryptographic suite.

An *Authenticate* command shall use the *in-process* reply specified in 6.3.1.6.3. The parameters that a Tag includes in its response are specified by the cryptographic suite. See Annex M.

Table 6.58 — *Authenticate* command

	Command	RFU	SenRep	IncRepLen	CSI	Length	Message	RN	CRC
# of bits	8	2	1	1	8	12	Variable	16	16
description	11010101	00	0: store 1: send	0: Omit length from reply 1: Include length in reply	CSI	length of message	message (depends on CSI)	handle	CRC-16

If a Tag receives an *Authenticate* specifying an unsupported CSI, an improperly formatted or not-executable message, or an improper cryptographic parameter then the Tag shall not execute the *Authenticate* and instead treat the command's parameters as unsupported (see Table C.30). If a Tag in the **secured** state receives an *Authenticate* that begins a new authentication, such as if the *Authenticate* contains a changed CSI, then the Tag shall transition to the **open** state, discontinue using and reset the current cryptographic engine, and begin the new authentication.

If a Tag receives a properly formatted *Authenticate* but there is a cryptographic error, and the cryptographic suite specifies that the error requires a security timeout, then the Tag shall set a security timeout as specified in 6.3.2.5. If a Tag that supports security timeouts for the *Authenticate* command receives an *Authenticate* during a timeout then it shall reject the command, backscatter an error code (see Annex I), and remain in its current state.

6.3.2.12.3.11 *AuthComm* (optional)

Interrogators and Tags may implement the *AuthComm* command; if they do, they shall implement it as shown in Table 6.59. *AuthComm* allows authenticated communications from R=>T by encapsulating another command and typically also a MAC in the *AuthComm's* message field. Table 6.28 shows the commands that an *AuthComm* may encapsulate. The generic nature of an *AuthComm* allows it to support a wide variety of cryptographic suites. An *AuthComm* shall always be preceded by a Tag, Interrogator, or mutual authentication via an *Authenticate* or a *Challenge*. The cryptographic suite indicated by the CSI in the *Authenticate* or *Challenge* that preceded the *AuthComm* specifies message and reply formatting. A Tag may include a MAC in its reply, again as specified by the cryptographic suite. A Tag only executes an *AuthComm* in the **open** or **secured** state. *AuthComm* has the following fields:

— IncRepLen specifies whether the Tag omits or includes length in its reply. If IncRepLen=0 then the Tag omits length from its reply; if IncRepLen=1 then the Tag includes length in its reply.

— Message includes the encapsulated command and other parameters (such as a MAC) as specified by the cryptographic suite. An Interrogator shall remove the command's preamble, handle, and CRC before encapsulating it in an *AuthComm*. The encapsulated command shall not be encrypted or obscured.

An *AuthComm* contains 2 RFU bits. An Interrogator shall set these bits to 00_2. A Tag in the **open** or **secured** states that receives an *AuthComm* with nonzero RFU bits shall not execute the *AuthComm* and instead treat the command's parameters as unsupported (see Table C.30). Future protocols may use these RFU bits to expand the *AuthComm* command's functionality.

A Tag in the **open** or **secured** states that receives an *AuthComm* encapsulating a disallowed command, an unsupported command, or a command that does not support encapsulation (see Table 6.28) shall not execute the *AuthComm* and instead treat the command's parameters as unsupported (see Table C.30).

An Interrogator shall prepend an *AuthComm* with a frame-sync (see 6.3.1.2.8).

A Tag shall only accept an *AuthComm* after a successful cryptographic authentication. Because an *Access* command sequence is not a cryptographic authentication, a Tag that most recently entered the **secured** state via a successful *Access* command sequence shall not execute an *AuthComm* and instead treat the command's parameters as unsupported (see Table C.30).

When processing an *AuthComm* a Tag shall first perform the functions/analysis/state-change/error-handling for the *AuthComm* itself and then, if the *AuthComm* is successful, the functions/analysis/state-change/error-handling for the command encapsulated in the *AuthComm*'s message field. In some instances, such as when an *AuthComm* encapsulates an authenticated *Kill*, the Tag may change state in response to the encapsulated command even though it did not change state in response to the *AuthComm* itself.

A Tag shall reply to an *AuthComm* using the *in-process* reply specified in 6.3.1.6.3. The cryptographic suite shall specify the parameters that a Tag includes in its response, including at least the reply for the encapsulated command minus preamble, handle, and CRC. For example, if the encapsulated command is a *Read* then the reply includes at least the read data or an error code as appropriate for a *Read*. Unlike other commands that use an *in-process* reply, *AuthComm* does not include a SenRep field because a Tag shall always send (i.e. never store) its reply to an *AuthComm*.

An *AuthComm* may exhibit behavior different from other commands because an *AuthComm* itself may succeed or fail or the encapsulated command, such as a *Lock*, may succeed or fail. Done and header in the reply of Table 6.14 indicate success or failure of the *AuthComm*. Response in the reply of Table 6.14 indicates success or failure of the encapsulated command. For example, suppose a Tag receives an *AuthComm* with IncRepLen=1 encapsulating a command whose reply type is *delayed* (such as a *Lock*). Upon successfully completing the *Lock* the Tag's reply will be as shown in Table 6.14 with done=1 and header=0, indicating that the *AuthComm* executed successfully, and length=0002_h and result=0, indicating that the *Lock* completed successfully. Note that this example presumes that the Tag was in the **secured** state; if the Tag was in the **open** state then the *AuthComm* would succeed but the reply to the *Lock* would be an error code.

If a Tag receives a properly formatted *AuthComm* but there is a cryptographic error, and the cryptographic suite specifies that the error requires a security timeout, then the Tag shall set a security timeout as specified in 6.3.2.5. If a Tag that supports security timeouts for the *AuthComm* command receives an *AuthComm* during a timeout then it shall reject the command, backscatter an error code (see Annex I), and remain in its current state.

Table 6.59 — *AuthComm* command

	Command	RFU	IncRepLen	Message	RN	CRC
# of bits	8	2	1	Variable	16	16
description	11010111	00_2	0: Omit length from reply 1: Include length in reply	message	handle	CRC-16

6.3.2.12.3.12 *SecureComm* (optional)

Interrogators and Tags may implement the *SecureComm* command; if they do, they shall implement it as shown in Table 6.60. *SecureComm* allows encrypted communications from R=>T by encapsulating another, encrypted command in the *SecureComm's* message field. Table 6.28 shows the commands that a *SecureComm* may encapsulate. The generic nature of a *SecureComm* allows it to support a wide variety of cryptographic suites. A *SecureComm* shall always be preceded by a Tag, Interrogator, or mutual authentication via an *Authenticate* or a *Challenge*. The cryptographic suite indicated by the CSI in the *Authenticate* or *Challenge* that preceded the *SecureComm* specifies message and reply formatting. A Tag

may encrypt and/or include a MAC in its reply, again as specified by the cryptographic suite. A Tag only executes a *SecureComm* in the **open** or **secured** state. *SecureComm* has the following fields:

— SenRep specifies whether a Tag backscatters its response or stores the response in its ResponseBuffer.

— IncRepLen specifies whether the Tag omits or includes length in its reply. If IncRepLen=0 then the Tag omits length from its reply; if IncRepLen=1 then the Tag includes length in its reply.

— Length is the message length, in bits.

— Message includes the encapsulated command and other parameters (such as a MAC) as specified by the cryptographic suite. An Interrogator shall remove the command's preamble, handle, and CRC before encapsulating it in a *SecureComm*. The encapsulated command shall be encrypted.

A *SecureComm* contains 2 RFU bits. An Interrogator shall set these bits to 00_2. A Tag in the **open** or **secured** states that receives a *SecureComm* with nonzero RFU bits shall not execute the *SecureComm* and instead treat the command's parameters as unsupported (see Table C.30). Future protocols may use these RFU bits to expand the *SecureComm* command's functionality.

A Tag in the **open** or **secured** states that receives a *SecureComm* encapsulating a disallowed command, an unsupported command, or a command that does not support encapsulation (see Table 6.28) shall not execute the *SecureComm* and instead treat the command's parameters as unsupported (see Table C.30).

An Interrogator shall prepend a *SecureComm* with a frame-sync (see 6.3.1.2.8).

A Tag shall only accept a *SecureComm* after a successful cryptographic authentication. Because an *Access* command sequence is not a cryptographic authentication, a Tag that most recently entered the **secured** state via a successful *Access* command sequence shall not execute a *SecureComm* and instead treat the command's parameters as unsupported (see Table C.30).

When processing a *SecureComm* a Tag shall first perform the functions/analysis/state-change/error-handling for the *SecureComm* itself and then, if the *SecureComm* is successful, the functions/analysis/state-change/error-handling for the command encapsulated in the *SecureComm*'s message field. In some instances, such as when a *SecureComm* encapsulates an authenticated *Kill*, the Tag may change state in response to the encapsulated command even though it did not change state in response to the *SecureComm* itself.

A Tag shall reply to a *SecureComm* using the *in-process* reply specified in 6.3.1.6.3. The cryptographic suite shall specify the parameters that a Tag includes in its response, including at least the reply for the encapsulated command minus preamble, handle, and CRC. For example, if the encapsulated command is a *Read* then the reply includes at least the read data or an error code as appropriate for a *Read*.

A *SecureComm* may exhibit behavior different from other commands because a *SecureComm* itself may succeed or fail or the encapsulated command, such as a *Lock*, may succeed or fail. Done and header in the reply of Table 6.14 indicate success or failure of the *SecureComm*. Response in the reply of Table 6.14 indicates success or failure of the encapsulated command. For example, suppose a Tag receives a *SecureComm* with IncRepLen=1 and SenRep=1 encapsulating a command whose reply type is *delayed* (such as a *Lock*). Upon successfully completing the *Lock* the Tag's reply will be as in Table 6.14 with done=1 and header=0, indicating the *SecureComm* executed successfully, and length=0002_h and result=0, indicating that the *Lock* completed successfully. Alternatively, if SenRep=0 then the reply will be as shown in Table 6.14 with done=1 and header=0, indicating the *SecureComm* executed successfully, and length=0002_h and result=null, indicating that the *Lock* completed successfully and result (0_2) is in the ResponseBuffer. In this latter case the Tag asserts **C** in XPC_W1 to indicate that the ResponseBuffer contains a computed result. This example presumes that the Tag was in the **secured** state; if it was in the **open** state then the *SecureComm* would succeed but the reply to the *Lock* would be an error code.

If a Tag receives a properly formatted *SecureComm* but there is a cryptographic error, and the cryptographic suite specifies that the error requires a security timeout, then the Tag shall set a security timeout as specified in 6.3.2.5. If a Tag that supports security timeouts for the *SecureComm* command receives a *SecureComm* during a timeout then it shall reject the command, backscatter an error code (see Annex I), and remain in its current state.

Table 6.60 — *SecureComm* command

	Command	RFU	SenRep	IncRepLen	Length	Message	RN	CRC
# of bits	8	2	1	1	12	Variable	16	16
description	11010110	00	0: store 1: send	0: Omit length from reply 1: Include length in reply	length of message	message	handle	CRC-16

6.3.2.12.3.13 *KeyUpdate* (optional)

Interrogators and Tags may implement the *KeyUpdate* command; if they do, they shall implement it as shown in Table 6.61. *KeyUpdate* allows an Interrogator to write or overwrite a key stored in a Tag. The generic nature of a *KeyUpdate* allows it to support a wide variety of cryptographic suites. A *KeyUpdate* shall always be preceded by an Interrogator or mutual authentication via an *Authenticate*. The cryptographic suite indicated by the CSI in the *Authenticate* that preceded the *KeyUpdate* specifies message and reply formatting. A Tag only executes a *KeyUpdate* in the **secured** state. *KeyUpdate* has the following fields:

— SenRep specifies whether a Tag backscatters its response or stores the response in its ResponseBuffer.

— IncRepLen specifies whether the Tag omits or includes length in its reply. If IncRepLen=0 then the Tag omits length from its reply; if IncRepLen=1 then the Tag includes length in its reply.

— Length is the message length, in bits.

— KeyID specifies the key to be written or updated.

— Message is or contains the key. Message may contain other parameters (such as a MAC) as specified by the cryptographic suite. Message may be encrypted.

A *KeyUpdate* contains 2 RFU bits. An Interrogator shall set these bits to 00_2. A Tag in the **secured** state that receives a *KeyUpdate* with nonzero RFU bits shall not execute the *KeyUpdate* and instead treat the command's parameters as unsupported (see Table C.30). Future protocols may use these RFU bits to expand the *KeyUpdate* command's functionality.

An Interrogator may encapsulate a *KeyUpdate* in a *SecureComm* or an *AuthComm* (see Table 6.28). If a cryptographic suite requires that *KeyUpdate* be encapsulated in a *SecureComm* then message in the *KeyUpdate* need not be encrypted. If a cryptographic suite allows sending a *KeyUpdate* in an *AuthComm* or without encapsulation then message in the *KeyUpdate* shall be encrypted.

A Tag in the **secured** state shall only write a key if (a) the Interrogator authenticated itself as a crypto superuser and KeyID is assigned to the same cryptographic suite as that specified by CSI in the *Authenticate* command that preceded the *KeyUpdate*, or (b) KeyID is the same as that used by the Interrogator to authenticate itself. In all other instances the Tag shall not execute the *KeyUpdate* and instead treat the command's parameters as unsupported (see Table C.30). See 6.3.2.11.2 for a description of Tag privileges and the crypto superuser privilege.

Upon receiving an executable *KeyUpdate* a Tag shall overwrite its old key with the new key. If the Tag does not write the new key successfully then it shall revert to the prior stored key. A Tag may meet this latter requirement by, for example, double-buffering the write operation.

An Interrogator shall prepend an unencapsulated *KeyUpdate* with a frame-sync (see 6.3.1.2.8).

Table 6.61 — *KeyUpdate* command

	Command	RFU	SenRep	IncRepLen	KeyID	Length	Message	RN	CRC
# of bits	16	2	1	1	8	12	Variable	16	16
description	11100010 00000010	00	0: store 1: send	0: Omit length from reply 1: Include length in reply	KeyID	length of message	message	handle	CRC-16

A Tag shall only accept a *KeyUpdate* after a successful cryptographic authentication. Because an *Access* command sequence is not a cryptographic authentication, a Tag that most recently entered the **secured** state via a successful *Access* command sequence shall not execute a *KeyUpdate* and instead treat the command's parameters as unsupported (see Table C.30).

A Tag shall reply to a *KeyUpdate* using the *in-process* reply specified in 6.3.1.6.3. The cryptographic suite shall specify the parameters that a Tag includes in its response.

If a Tag receives a properly formatted *KeyUpdate* but there is a cryptographic error, and the cryptographic suite specifies that the error requires a security timeout, then the Tag shall set a security timeout as specified in 6.3.2.5. If a Tag that supports security timeouts for the *KeyUpdate* command receives a *KeyUpdate* during a timeout then it shall reject the command, backscatter an error code (see Annex I), and remain in its current state.

6.3.2.12.3.14 TagPrivilege (optional)

Interrogators and Tags may implement the *TagPrivilege* command; if they do, they shall implement it as shown in Table 6.62. *TagPrivilege* allows an Interrogator to read or modify the Tag privileges in Table 6.22 or Table 6.23 for the access password or for a key, respectively. A Tag only executes a *TagPrivilege* in the **secured** state. *TagPrivilege* has the following fields:

— SenRep specifies whether a Tag backscatters its response or stores the response in its ResponseBuffer.

— IncRepLen specifies whether the Tag omits or includes length in its reply. If IncRepLen=0 then the Tag omits length from its reply; if IncRepLen=1 then the Tag includes length in its reply.

— Action specifies whether the Interrogator is reading privileges or modifying them. Action=0 indicates read; Action=1 indicates modify.

— Target specifies whether the Interrogator is targeting the access password or a key. If Target=0 then the Tag reads or modifies the access-password privileges; if Target=1 then the Tag reads or modifies the Tag privileges for the key indicated by KeyID.

— KeyID specifies the key for the privileges being read or written.

— Privilege specifies values for each of the 16 Tag privileges in Table 6.22 or Table 6.23 when an Interrogator is modifying a privilege (i.e. when Action=1).

A *TagPrivilege* contains 2 RFU bits. An Interrogator shall set these bits to 00_2. A Tag in the **secured** state that receives a *TagPrivilege* containing nonzero RFU bits shall not execute the *TagPrivilege* and instead treat the command's parameters as unsupported (see Table C.30). Future protocols may use these RFU bits to expand the *TagPrivilege* command's functionality.

An unauthenticated Interrogator may issue a *TagPrivilege*; if it does then it shall issue the *TagPrivilege* without encapsulation and with Target=0 (i.e. specifying the access password).

An authenticated Interrogator shall encapsulate a *TagPrivilege* in a *SecureComm* or *AuthComm* (see Table 6.28). If a Tag in the **secured** state receives an unencapsulated *TagPrivilege* from an authenticated Interrogator then it shall not execute the *TagPrivilege* and instead treat the command's parameters as unsupported (see Table C.30).

A Tag in the **secured** state shall only read or modify the access-password privileges if the Interrogator supplied the correct access password and is not attempting to assert a deasserted privilege. In all other instances the Tag shall not execute the *TagPrivilege* and instead treat the command's parameters as unsupported (see Table C.30).

A Tag in the **secured** state shall only read or modify a key's privileges if (a) the Interrogator authenticated itself as a crypto superuser and KeyID is assigned to the same cryptographic suite as that specified by CSI in the *Authenticate* command that preceded the *TagPrivilege*, or (b) KeyID is the same as that used by the Interrogator to authenticate itself and the Interrogator is not attempting to assert a deasserted privilege.

If an Interrogator specifies Action=0 in a *TagPrivilege* then it may use any value for privilege. A Tag shall ignore privilege when Action=0.

If an Interrogator specifies Target=0 in a *TagPrivilege* then it may use any value for the KeyID. If Tag receives a *TagPrivilege* with Target=0 then it shall ignore the value that the Interrogator supplies for KeyID.

Upon receiving an executable *TagPrivilege* with Action=1 a Tag shall overwrite the old privileges with the new privileges. If the Tag does not write the new privileges successfully then it shall revert to the prior stored privileges. A Tag may meet this latter requirement by, for example, double-buffering the write operation.

A Tag in the **secured** state that receives a *TagPrivilege* which attempts to assert one or more RFU privilege bits or to change an unchangeable privilege value shall not execute the *TagPrivilege* and instead treat the command's parameters as unsupported (see Table C.30).

An Interrogator shall prepend an unencapsulated *TagPrivilege* with a frame-sync (see 6.3.1.2.8).

A Tag shall reply to a *TagPrivilege* using the *in-process* reply specified in 6.3.1.6.3. The Tag's response shall be as shown in Table 6.63 for Action=0 or Action=1. The response includes Target, the Interrogator-supplied KeyID, and the current privileges (newly written if Action=1 and the Tag wrote the new privileges successfully).

Table 6.62 — *TagPrivilege* command

	Command	RFU	SenRep	IncRepLen	Action	Target	KeyID	Privi-leges	RN	CRC
# of bits	16	2	1	1	1	1	8	16	16	16
description	11100010 00000011	00	0: store 1: send	0: Omit length from reply 1: Include length in reply	0: read 1: mod-ify	0: access pwd 1: key	KeyID	privilege	handle	CRC-16

Table 6.63 — Tag reply to a successful *TagPrivilege* command

	Target	KeyID	Privileges
# of bits	1	8	16
description	0: access pwd 1: key	KeyID	privilege

6.3.2.12.3.15 *ReadBuffer* (optional)

Interrogators and Tags may implement the *ReadBuffer* command; if they do, they shall implement it as shown in Table 6.64. *ReadBuffer* allows an Interrogator to read data stored in a Tag's ResponseBuffer.

A Tag only executes a *ReadBuffer* in the **open** or **secured** state and only if the Tag's **C** flag is asserted. *ReadBuffer* has the following fields:

— WordPtr specifies the starting word address for the read. For example, WordPtr=000_h specifies the first 16-bit memory word, WordPtr=001_h specifies the second 16-bit memory word, etc.

— BitCount specifies the number of bits to read. If BitCount=000_h then a Tag shall backscatter the contents of the ResponseBuffer starting at WordPtr and ending at the end of the allocated ResponseBuffer.

A *ReadBuffer* contains 2 RFU bits. An Interrogator shall set these bits to 00_2. A Tag in the **open** or **secured** states that receives a *ReadBuffer* with nonzero RFU bits shall not execute the *ReadBuffer* and instead treat the command's parameters as unsupported (see Table C.30). Future protocols may use these RFU bits to expand the *ReadBuffer* command's functionality.

Table 6.64 — *ReadBuffer* command

	Command	RFU	WordPtr	BitCount	RN	CRC
# of bits	8	2	12	12	16	16
description	11010010	00	Starting address pointer	Number of bits to read	handle	CRC-16

An Interrogator may encapsulate a *ReadBuffer* in an *AuthComm* but shall not encapsulate it in a *SecureComm* (see Table 6.28).

If a Tag implements a ResponseBuffer then that Tag shall implement the *ReadBuffer* command.

An Interrogator shall prepend an unencapsulated *ReadBuffer* with a frame-sync (see 6.3.1.2.8).

Table 6.65 — Tag reply to a successful *ReadBuffer* command

	Header	Data Bits	RN	CRC
# of bits	1	Variable	16	16
description	0	data	handle	CRC-16

A Tag shall reply to a *ReadBuffer* using the *immediate* reply specified in 6.3.1.6.1. If **C**=1 and the memory bits specified in the *ReadBuffer* exist then the Tag's reply shall be as shown in Table 6.65 including a header (a 0-bit), the data bits, and the Tag's handle. The reply includes a CRC-16 calculated over the 0-bit, data bits, and handle. If one or more of the memory bits specified in the *ReadBuffer* do not exist, or if the **C** flag in XPC_W1 is zero-valued, then the Tag shall not execute the *ReadBuffer* and instead backscatter an error code (see Table C.30, unsupported parameters) within time T_1 in Table 6.16 rather than the reply shown in Table 6.65.

6.3.2.12.3.16 *Untraceable* (optional)

Interrogators and Tags may implement the *Untraceable* command; if they do, they shall implement it as shown in Table 6.66. *Untraceable* allows an Interrogator with an asserted Untraceable privilege to instruct a Tag to (a) alter the **L** and **U** bits in UII memory, (b) hide memory from Interrogators with a deasserted Untraceable privilege, and/or (c) reduce its operating range for all Interrogators. The memory that a Tag may hide includes words of UII memory, the Tag serialization in TID memory, all of TID memory, and/or User memory (File_0 and above). Untraceable and traceable Tags behave identically from a state-machine and command-response perspective; the difference between them is (a) the memory the Tag exposes to an Interrogator with a deasserted Untraceable privilege and/or (b) the Tag's operating range. A Tag only executes an *Untraceable* in the **secured** state. *Untraceable* has the following fields:

— U specifies a value for the **U** bit in XPC_W1 (see 6.3.2.1.2.2). Upon receiving an *Untraceable* command a Tag that supports the **U** bit shall overwrite bit $21C_h$ of XPC_W1 with the provided U value regardless

of the lock or permalock status of UII memory. If the Tag does not support the **U** bit then the Tag shall ignore the provided U value but continue to process the remainder of the *Untraceable*.

— UII includes a show/hide bit (MSB) and 5 length bits (5 LSBs). These fields operate independently.

  — Show/hide specifies whether a Tag untraceably hides part of UII memory. If show/hide=0_2 then a Tag exposes UII memory. If show/hide=1_2 then a Tag untraceably hides UII memory above that set by its UII length field (i.e. StoredPC bits 10_h – 14_h) to bit $20F_h$ (inclusive).

  — Length specifies a new UII length field (**L** bits). Upon receiving an *Untraceable* command a Tag shall overwrite its UII length field (StoredPC bits 10_h – 14_h) with the provided length bits regardless of the lock or permalock status of UII memory. In response to subsequent *ACK*s the Tag backscatters a UII whose length is set by the new length bits.

— TID specifies the TID memory that a Tag untraceably hides. If TID=00_2 then a Tag exposes TID memory. If TID=01_2 and a Tag's allocation class identifier (see 6.3.2.1.3) is $E0_h$ then the Tag untraceably hides TID memory above 10_h, inclusive; if the Tag's allocation class identifier is $E2_h$ then the Tag untraceably hides TID memory above 20_h, inclusive. If TID=10_2 then the Tag untraceably hides all of TID memory. TID=11_2 is RFU.

— User specifies whether a Tag untraceably hides User memory. If User=0_2 then the Tag exposes User memory. If User=1_2 then the Tag untraceably hides User memory (i.e. hides File_0 and above).

— Range specifies a Tag's operating range. If range=00_2 then the Tag persistently enables normal operating range. If range=10_2 then the Tag persistently enables reduced operating range. If range=01_2 then the Tag temporarily toggles its operating range (if normal then to reduced; if reduced then to normal) but reverts to its prior persistent operating range when the Tag loses power. Temporary toggling allows an Interrogator to confirm that a Tag is still readable before committing range-reduced untraceability to the Tag's nonvolatile memory (by sending a subsequent *Untraceable* with range=10_2). Range=11_2 is RFU. A Tag shall execute a range change prior to replying to the *Untraceable*. The range-reduction details, including its magnitude and the commands to which it applies, are manufacturer-defined. If a Tag does not support range reduction then it shall ignore range but continue to process the remainder of the *Untraceable*.

An *Untraceable* contains 2 RFU bits. An Interrogator shall set these bits to 00_2. A Tag in the **secured** state that receives an *Untraceable* with nonzero RFU bits, TID=11_2, or range=11_2 shall not execute the *Untraceable* and instead treat the command's parameters as unsupported (see Table C.30). Future protocols may use these RFU bits to expand the *Untraceable* command's functionality.

If a Tag in the **secured** state receives an *Untraceable* from an Interrogator with an asserted Untraceable privilege then it shall execute the command; if the Interrogator has a deasserted Untraceable privilege then the Tag shall not execute the command and instead treat the command's parameters as unsupported (see Table C.30).

An unauthenticated Interrogator may issue an *Untraceable* without encapsulation. An authenticated Interrogator shall encapsulate an *Untraceable* in a *SecureComm* or *AuthComm* (see Table 6.28). If a Tag in the **secured** state receives an unencapsulated *Untraceable* from an authenticated Interrogator then it shall not execute the *Untraceable* and instead treat the command's parameters as unsupported (see Table C.30).

Untraceable commands shall be atomic, meaning that a Tag, upon receiving an executable *Untraceable*, shall discard its prior memory and range settings and implement the new ones.

If an *Untraceable* command modifies a Tag's UII length field and the Tag computes its StoredCRC at powerup then the StoredCRC is likely to be incorrect until the Interrogator power-cycles the Tag. See 6.3.2.1.2.1.

If a Tag supports only **XI**=0_2 then the length bits in an *Untraceable* may have any 5-bit value. If the Tag supports **XI**=1_2 then the maximum length-bit value is 11101_2. A Tag that supports **XI**=1_2 shall not execute an *Untraceable* that specifies length bits greater than 11101_2 and shall instead treat the command's parameters as unsupported (see Table C.30). Regardless of these absolute bounds on length,

if an *Untraceable* specifies a length value that a Tag does not support then the Tag shall not execute the *Untraceable* and instead treat the command's parameters as unsupported (see Table C.30).

A Tag that is operating with reduced range shall do so for all commands regardless of whether an Interrogator has an asserted or a deasserted Untraceable privilege.

A Tag shall execute supported access commands that operate on untraceably hidden memory if the commanding Interrogator has an asserted Untraceable privilege, but shall not execute these commands if the Interrogator has a deasserted Untraceable privilege. In the latter case a Tag shall behave as though untraceably hidden memory does not exist and treat the commands' parameters as unsupported (see Table C.30). As an example, suppose that a Tag's User memory is untraceably hidden. The Tag may execute a *FileOpen* from an Interrogator with an asserted Untraceable privilege but not from an Interrogator with a deasserted Untraceable privilege.

A Tag that is untraceably hiding UII memory shall not include any of the untraceably hidden UII memory bits when replying to an *ACK*.

This protocol recommends that an Interrogator permalock the UII memory bank prior to untraceably hiding part or all of UII memory. Absent such permalocking, an Interrogator without the Untraceable privilege may subsequently alter the Tag's UII length field and expose untraceably hidden memory.

A Tag treats as not-matching a *Select* command whose Mask includes untraceably hidden memory.

If a Tag computes its **UMI** then the untraceability status of User memory does not change the **UMI** value.

An Interrogator shall prepend an unencapsulated *Untraceable* with a frame-sync (see 6.3.1.2.8).

This protocol allows a Tag manufacturer to implement irreversible untraceability whereby a memory region, once untraceably hidden, cannot be re-exposed and/or the Tag's operating range, once reduced, cannot be restored to normal. The details of this irreversible untraceability, including whether a Tag with irreversibly hidden memory will still alter its operating range, and vice versa, shall be Tag-manufacturer defined.

Table 6.66 — *Untraceable* command

	Command	RFU	U	UII	TID	User	Range	RN	CRC
# of bits	16	2	1	6	2	1	2	16	16
description	11100010 00000000	00	0: Deassert **U** in XPC_W1 1: Assert **U** in XPC_W1	MSB (show/hide): 0: show memory above UII 1: hide memory above UII 5 LSBs (length): New UII length field (new **L** bits)	00: hide none 01: hide some 10: hide all 11: RFU	0: view 1: hide	00: normal 01: toggle temporarily 10: reduced 11: RFU	handle	CRC-16

This protocol allows a Tag manufacturer to configure a Tag to only execute an *Untraceable* at short range. This protocol also allows a Tag manufacturer to configure such a Tag with a zero-valued access password and an asserted Untraceable privilege for the access password, in which case the short-range feature provides the only protection against illicit use of the *Untraceable* command.

A Tag shall reply to an *Untraceable* using the *delayed* reply specified in 6.3.1.6.2. Upon receiving an executable *Untraceable* a Tag shall perform the specified actions. If a Tag receives an *Untraceable* whose fields it supports but nonetheless cannot execute, such as if the *Untraceable* instructs the Tag to expose an irreversibly hidden portion of Tag memory or the Interrogator has a deasserted Untraceable privilege, then the Tag shall not execute the *Untraceable* and instead treat the command's parameters as unsupported (see Table C.30).

6.3.2.12.3.17 *FileOpen* (optional)

Interrogators and Tags may implement the *FileOpen* command; if they do, they shall implement it as shown in Table 6.67. *FileOpen* allows an Interrogator to instruct a Tag to close the currently open file and open a new file. A Tag only executes a *FileOpen* in the **open** or **secured** state. *FileOpen* has the following fields:

— FileNum specifies the file to be opened.

A *FileOpen* contains 2 RFU bits. An Interrogator shall set these bits to 00_2. A Tag in the **open** or **secured** states that receives a *FileOpen* with nonzero RFU bits or that specifies FileNum=11111111_2 (RFU FileNum) shall not execute the *FileOpen* and instead treat the command's parameters as unsupported (see Table C.30). Future protocols may use these RFU bits to expand the *FileOpen* command's functionality

An authenticated Interrogator shall encapsulate a *FileOpen* in a *SecureComm* or *AuthComm* (see Table 6.28). If a Tag in the **secured** state receives an unencapsulated *FileOpen* from an authenticated Interrogator then it shall not execute the *FileOpen* and instead treat the command's parameters as unsupported (see Table C.30).

An unauthenticated Interrogator may issue a *FileOpen* without encapsulation to open files accessible from a Tag (a) in the **open** state, or (b) in the **secured** state by an Interrogator that supplied the access password.

Table 6.67 — *FileOpen* command

	Command	RFU	FileNum	RN	CRC
# of bits	8	2	10	16	16
description	11010011	00	Which file to open	handle	CRC-16

If an Interrogator or a Tag support File_N, N>0 then that Interrogator or Tag shall implement a *FileOpen*.

If an Interrogator issues a *FileOpen* specifying a File_N, N>0 for which the Interrogator has a 0000_2 file privilege value then the Tag will open the file, but the Interrogator will not be able to read any data in or otherwise modify the file (see Table 6.24 and Table 6.25).

An Interrogator shall prepend an unencapsulated *FileOpen* with a frame-sync (see 6.3.1.2.8).

If a Tag supports the *FileOpen* command then it shall implement the file (**F**) indicator (see 6.3.2.1.3).

Table 6.68 — Tag reply to a successful *FileOpen* command

	Header	FileNum	FileType	FileSize	BlockSize	IntPriv	LastFile	RN	CRC
# of bits	1	10	8	10	10	4	1	16	16
description	0	Open file	FileType	File size in blocks	Block size in words	Interrogator's file privilege	0: Not max FileNum 1: Max FileNum	handle	CRC-16

A Tag shall reply to a *FileOpen* using the *immediate* reply specified in 6.3.1.6.1. If the Tag has an allocated file at FileNum then it shall close the currently open file, open the specified file, and reply as shown in Table 6.68. The reply includes a header (a 0-bit), FileNum, FileType, FileSize, BlockSize, IntPriv, LastFile, and the Tag's handle. FileNum, FileType, FileSize, BlockSize are defined in 6.3.2.11.3. IntPriv is the Interrogator's 4-bit privilege to the file (see Table 6.24 and Table 6.25). LastFile indicates whether the just-opened file has the largest assigned FileNum; if a Tag has a FileNum larger than that of the just-opened file then it shall set LastFile to 0, otherwise it shall set LastFile to 1. The reply includes a CRC-16 calculated over the 0-bit to the last handle bit. If a Tag receives a *FileOpen* specifying the currently open file then it shall leave the file open and reply as specified in Table 6.68. If a Tag receives a *FileOpen* but does not have an allocated file at FileNum, or if User memory is untraceably hidden and the Interrogator has a deasserted Untraceable privilege, or if the Tag is otherwise unable to execute the *FileOpen*, then

the Tag shall not execute the *FileOpen* and instead treat the command's parameters as unsupported (see Table C.30), reverting to the currently open file (or to no file if the Tag doesn't have any allocated files or if User memory is untraceably hidden and the Interrogator has a deasserted Untraceable privilege).

6.3.2.12.3.18 *FileList* (optional)

Interrogators and Tags may implement the *FileList* command; if they do, they shall implement it as shown in Table 6.69. *FileList* allows an Interrogator to obtain information about a Tag's files and the Interrogator's privileges to those files. A Tag only executes a *FileList* in the **open** or **secured** state. *FileList* has the following fields:

— SenRep specifies whether a Tag backscatters its response or stores the response in its ResponseBuffer.

— IncRepLen specifies whether the Tag omits or includes length in its reply. If IncRepLen=0 then the Tag omits length from its reply; if IncRepLen=1 then the Tag includes length in its reply.

— FileNum identifies the starting file for which the Interrogator is requesting information, inclusive.

— AddlFiles identifies the number of additional files for which the Interrogator is requesting information. For example, if FileNum=4 and AddlFiles=2 then the Tag shall provide information for File_4 and for the next two higher-numbered files (which may be File_5 and File_6 if the Tag manufacturer assigned file numbers sequentially or may be other files if the numbering is not sequential).

A *FileList* contains 2 RFU bits. An Interrogator shall set these bits to 00_2. A Tag in the **open** or **secured** states that receives a *FileList* with nonzero RFU bits or that specifies FileNum=1111111111_2 (RFU FileNum) shall not execute the *FileList* and instead treat the command's parameters as unsupported (see Table C.30). Future protocols may use these RFU bits to expand the *FileList* command's functionality.

An authenticated Interrogator shall encapsulate a *FileList* in a *SecureComm* or *AuthComm* (see Table 6.28). If a Tag in the **secured** state receives an unencapsulated *FileList* from an authenticated Interrogator then it shall not execute the *FileList* and instead treat the command's parameters as unsupported (see Table C.30).

Table 6.69 — *FileList* command

	Command	RFU	SenRep	IncRepLen	FileNum	AddlFiles	RN16	CRC
# of bits	16	2	1	1	10	8	16	16
description	11100010 00000001	00	0: store 1: send	0: Omit length from reply 1: Include length in reply	First file number	Total number of additional files	handle	CRC-16

An unauthenticated Interrogator may issue a *FileList* without encapsulation to a Tag in the **open** or **secured** state.

An Interrogator shall not specify AddlFiles=FF_h. If a Tag receives a *FileList* with AddlFiles=FF_h then the Tag shall behave as though it had received a *FileList* with AddlFiles=FD_h.

An Interrogator shall prepend an unencapsulated *FileList* with a frame-sync (see 6.3.1.2.8).

Table 6.70 — Tag reply to a successful *FileList* command

	NumMessages	Message 1	...	Message N	BlockSize	AvailFileSize
# of bits	8	32	...	32	10	10
description	Number of messages in this reply	[FileNum, FileType, FileSize, IntPriv]	...	[FileNum, FileType, FileSize, IntPriv]	Block size in words	Allocateable memory in blocks

A Tag shall reply to a *FileList* using the *in-process* reply specified in 6.3.1.6.3. A Tag's response shall be as shown in Table 6.70 and includes a message for each file for which the Interrogator requested

information. The response includes the number of messages, the message contents (10-bit FileNum, 8-bit FileType, 10-bit FileSize, 4-bit IntPriv), the BlockSize, and the free memory available for file resizing (AvailFileSize). IntPriv is the Interrogator's 4-bit privilege to the file (see Table 6.24 and Table 6.25). If a Tag is *static* then AvailFileSize shall be zero. If a Tag has more than 1022 blocks of free memory then AvailFileSize shall be 1111111111_2. If a Tag receives a *FileList* with an unsupported FileNum, or AddlFiles exceeds the number of files above FileNum, or User memory is untraceably hidden and the Interrogator has a deasserted Untraceable privilege, or the Tag is otherwise unable to execute the *FileList*, then the Tag shall not execute the *FileList* and instead treat the command's parameters as unsupported (see Table C.30).

6.3.2.12.3.19 FilePrivilege (optional)

Interrogators and Tags may implement the *FilePrivilege* command; if they do, they shall implement it as shown in Table 6.72. *FilePrivilege* allows an Interrogator to read or modify file privileges (Table 6.24 or Table 6.25) for the currently open file. A Tag only executes a *FilePrivilege* in the **secured** state. *FilePrivilege* has the following fields:

— SenRep specifies whether a Tag backscatters its response or stores the response in its ResponseBuffer.

— IncRepLen specifies whether the Tag omits or includes length in its reply. If IncRepLen=0 then the Tag omits length from its reply; if IncRepLen=1 then the Tag includes length in its reply.

— Action specifies whether the Interrogator is reading or modifying a privilege for the currently open file, and if modifying whether the change applies to the **open** state, access password, a single key, or all keys.

— KeyID specifies a key.

— Privilege specifies the file privilege. See Table 6.24 and Table 6.25.

A *FilePrivilege* contains 2 RFU bits. An Interrogator shall set these bits to 00_2. A Tag in the **secured** state that receives a *FilePrivilege* with nonzero RFU bits shall not execute the *FilePrivilege* and instead treat the command's parameters as unsupported (see Table C.30). Future protocols may use these RFU bits to expand the *FilePrivilege* command's functionality.

An authenticated Interrogator shall encapsulate a *FilePrivilege* in a *SecureComm* or *AuthComm* (see Table 6.28). If a Tag in the **secured** state receives an unencapsulated *FilePrivilege* from an authenticated Interrogator then it shall not execute the *FilePrivilege* and instead treat the command's parameters as unsupported (see Table C.30).

An unauthenticated Interrogator may issue a *FilePrivilege* to a Tag in the **secured** state without encapsulation.

A Tag shall execute a *TagPrivilege* according to Table 6.71 which specifies, for each Action value, the privilege assignment that the Tag makes (if any), the fields in the *FilePrivilege* that the Tag ignores, the required Tag or file privilege to perform the requested operation, and the reply that the Tag backscatters. An Interrogator may set an ignored field in a *FilePrivilege* to any value.

Table 6.71 — Action field behavior for a *FilePrivilege*

Action	Privilege a Tag Assigns to the Currently Open File	Tag Ignores	Required Privilege	Tag Reply to the *FilePrivilege*	Reference
000_2	–	KeyID and privilege	Any	FileNum and the **open**-state file privilege	Table 6.73, Action=000_2 or 001_2
001_2	privilege to the **open** state for FileNum	KeyID	File superuser	–	

Table 6.71 *(continued)*

Action	Privilege a Tag Assigns to the Currently Open File	Tag Ignores	Required Privilege	Tag Reply to the *FilePrivilege*	Reference
010_2	–	KeyID and privilege	Any	FileNum and the access-password file privilege	Table 6.73, Action=010_2 or 011_2
011_2	privilege to the access password for FileNum	KeyID	— File superuser to assign privilege for the access password — DecFilePriv to decrement privilege for supplied access password	–	
100_2	–	privilege	Any	FileNum, KeyID, and KeyID's file privilege	Table 6.73, Action=100_2 or 101_2
101_2	privilege to KeyID for FileNum	–	— File superuser to assign privilege for any Key_N — DecFilePriv to decrement privilege for supplied key	–	
110_2	–	KeyID and privilege	Any	FileNum, NumKeys, and a KeyID/privilege pair for each key	Table 6.73, Action=110_2 or 111_2
111_2	privilege to all KeyID for FileNum	KeyID	File superuser	–	

As shown in Table 6.71, a Tag permits an Interrogator that is a file superuser to modify a file privilege for the **open** state, access password, or any key regardless of the cryptographic suite to which the key is assigned, for the currently open file. Table 6.71 also shows that a Tag permits an Interrogator that is not a file superuser to decrement the file privilege for the access password or key that it used to most recently enter the **secured** state if the access password or key has an asserted DecFilePriv, but only for the currently open file and not for the **open** state or for any other password or key. Finally, Table 6.71 shows that a Tag permits any Interrogator to read the privileges for the currently open file, for the **open** state, access password, or for any key.

Upon receiving an executable *FilePrivilege* with Action=001_2, 011_2, 101_2, or 111_2 a Tag shall overwrite the current file privilege(s) with the new privilege. If the Tag does not write the new privilege successfully then it shall revert to the prior stored privilege. A Tag may meet this latter requirement by, for example, double-buffering the write operation. Note that, if Action=111_2 then a Tag may complete the write operation for some keys but not for others, in which case an Interrogator can read the privilege values and, if necessary, re-issue a *FilePrivilege*.

Table 6.72 — *FilePrivilege* command

	Command	RFU	SenRep	IncRepLen	Action	KeyID	Privi-lege	RN	CRC
# of bits	16	2	1	1	3	8	4	16	16
description	11100010 00000100	00	0: store 1: send	0: Omit length from reply 1: Include length in reply	000: Read **open** state 001: Modify **open** state 010: Read access pwd 011: Modify access pwd 100: Read KeyID 101: Modify KeyID 110: Read all keys 111: Modify all keys	KeyID	privilege	han-dle	CRC-16

An Interrogator shall prepend an unencapsulated *FilePrivilege* with a frame-sync (see 6.3.1.2.8).

A Tag's response to the *FilePrivilege*, for incorporation into the *in-process* reply specified in 6.3.1.6.3, shall be as shown in Table 6.73.

Table 6.73 — Tag reply to a successful *FilePrivilege* with indicated Action fields

Action=000_2 or 001_2		
	FileNum	**Privilege**
# of bits	10	4
description	Currently open file	**open** state privilege

Action=010_2 or 011_2		
	FileNum	**Privilege**
# of bits	10	4
description	Currently open file	access password privilege

Action=100_2 or 101_2			
	FileNum	**KeyID**	**Privilege**
# of bits	10	8	4
description	Currently open file	KeyID	KeyID privilege

Action=110_2 or 111_2					
	FileNum	**NumKeys**	**Key_0 / Privilege_0 pair**	**...**	**Key_N / Privilege_N pair**
# of bits	10	8	12	...	12
description	Currently open file	Number of keys	Key_0 | Privilege_0	...	Key_N | Privilege_N

If a Tag receives a *FilePrivilege* that it cannot execute because the access password or key the Interrogator supplied has insufficient privileges, or the *FilePrivilege* contains an unsupported KeyID, or privilege is an RFU value, or User memory is untraceably hidden and the Interrogator has a deasserted Untraceable privilege, or the Tag is otherwise unable to execute the *FilePrivilege*, then the Tag shall not execute the *FilePrivilege* and instead treat the command's parameters as unsupported (see Table C.30).

6.3.2.12.3.20 *FileSetup* (optional)

Interrogators and Tags may implement the *FileSetup* command; if they do, they shall implement it as shown in Table 6.74. *FileSetup* allows an Interrogator to resize the currently open file, change its FileType, or both. A Tag only executes a *FileSetup* in the **secured** state. *FileSetup* has the following fields:

— SenRep specifies whether a Tag backscatters its response or stores the response in its ResponseBuffer.

— IncRepLen specifies whether the Tag omits or includes length in its reply. If IncRepLen=0 then the Tag omits length from its reply; if IncRepLen=1 then the Tag includes length in its reply.

— FileType specifies the new file type.

— FileSize specifies the requested file size in blocks.

A *FileSetup* contains 2 RFU bits. An Interrogator shall set these bits to 00_2. A Tag in the **secured** state that receives a *FileSetup* with nonzero RFU bits shall not execute the *FileSetup* and instead treat the command's parameters as unsupported (see Table C.30). Future protocols may use these RFU bits to expand the *FileSetup* command's functionality.

A Tag shall only execute a *FileSetup* issued by an Interrogator with a file superuser privilege (see 6.3.2.11.3).

An authenticated Interrogator shall encapsulate a *FileSetup* in a *SecureComm* or *AuthComm* (see Table 6.28). If a Tag in the **secured** state receives an unencapsulated *FileSetup* from an authenticated Interrogator then it shall not execute the *FileSetup* and instead treat the command's parameters as unsupported (see Table C.30).

An unauthenticated Interrogator may issue a *FileSetup* to a Tag in the **secured** state without encapsulation.

A *static* Tag that supports the *FileSetup* command shall permit an Interrogator with the file superuser privilege to modify a file's type but never its size. A *static* Tag shall write the FileType in a *FileSetup* as the file's new type and shall ignore FileSize. An Interrogator may set FileSize to any value when communicating with a *static* Tag.

A *dynamic* Tag shall permit an Interrogator with the file superuser privilege to modify a file's type and size. Table 6.26 specifies the conditions under which a *dynamic* Tag may be able to resize a file. When increasing a file's size a *dynamic* Tag shall only allocate "free" memory (i.e. memory not currently allocated to another file) to the resized file. When reducing a file's size a *dynamic* Tag may or may not, depending on the Tag manufacturer's implementation, erase the excised memory. Consequently, this protocol recommends that Interrogators, before reducing a file's size, erase that portion of the file that will be excised by the resizing. Whether a *dynamic* Tag is able to recover memory freed by resizing a file's size downward depends on the Tag manufacturer's implementation and is not specified by this protocol.

Regardless of whether a Tag is *static* or *dynamic*, after executing a *FileSetup* a Tag's response shall include both FileType and FileSize (even if the Tag made no changes to either one). See Table 6.75.

An Interrogator shall prepend an unencapsulated *FileSetup* with a frame-sync (see 6.3.1.2.8).

A Tag's response to the *FileSetup*, for incorporation into the *in-process* reply specified in 6.3.1.6.3, shall be as shown in Table 6.75. The response includes the FileNum, FileType, and FileSize. If a Tag receives a *FileSetup* that it cannot execute because the access password or key that the Interrogator most recently supplied does not have a file superuser privilege, or User memory is untraceably hidden and the Interrogator has a deasserted Untraceable privilege, or the Tag is otherwise unable to execute the

FileSetup, then the Tag shall not execute the *FileSetup* and instead treat the command's parameters as unsupported (see Table C.30).

There are many reasons why a *dynamic* Tag may be unable to execute a *FileSetup* including (a) the Tag does not have free memory to increase the file size, (b) the Tag has free memory but is unable to allocate it to the file, (c) the file has a permalocked block, and (d) many others. If a *dynamic* Tag is unable to execute the FileSize in the *FileSetup* command then it shall not execute any portion of the *FileSetup* (i.e. it shall not change the FileType) and instead treat the command's parameters as unsupported (see Table C.30).

Table 6.74 — *FileSetup* command

	Command	RFU	SenRep	IncRepLen	FileType	FileSize	RN16	CRC
# of bits	16	2	1	1	8	10	16	16
description	11100010 00000101	00	0: store 1: send	0: Omit length from reply 1: Include length in reply	FileType	Requested file size, in blocks	handle	CRC-16

Table 6.75 — Tag reply to a successful *FileSetup* command [1]

	FileNum	FileType	FileSize
# of bits	10	8	10
description	Currently open file	FileType	Current file size, in blocks

1: See also Table 6.26.

7 Battery Assisted Passive (BAP) Interrogator Talks First Type C systems (optional)

7.1 Applicability

In case an Interrogator or Tag supports any command, response or feature of Clause 7 then this Interrogator or Tag shall support all mandatory commands, responses or features, and it may support the specified combinations of optional commands, responses or features. If an Interrogator or Tag does not support any command, response, or feature of Clause 7, then Clause 7 does not apply for that device.

There are no new mandatory commands in Clause 7 simply to have a battery assisted Tag, but there are required modifications of flag persistence with batteries, and required relationships between optional commands and features.

7.2 General overview, definitions, and requirements of BAP

The benefit of BAP is to improve the robustness and performance of the air interface of a passive backscatter system. Specific benefits include better Tag sensitivity and resistance to multipath fade. These improvements can result in improved operating range or better penetration of reading zone into closely packed objects such as a pallet of goods. BAP also provides power for sensors which may then operate out of the field of an Interrogator. See Annex Q for an informative tutorial overview of BAP as implemented in this standard.

This clause defines the requirements and features of Battery Assisted Passive Type C systems. It covers these types of BAP Tags:

BAP PIE: The air interface of Clause 6 supplemented with an on-Tag battery and optionally supplemented with battery life extending Battery Saver functionality referred to as Battery Saver Mode. This BAP PIE mode may be provided as the only mode a Tag supports. Battery Saver Mode is basically a duty cycled mode of manufacturer selected form where the Tag receiver is either OFF or in

a very low power state, or switches between both, for the majority of the time. The Battery Saver Mode option may be implemented with a low power RF threshold detection function or alternatively with a successful command decode function as the trigger to activate the Tag and temporarily hold it in a fully operational state. Battery Saver Mode may feature either a low power continuous "**listen**" capability, or a duty cycled **listen** capability (typically at improved sensitivity compared to continuous), or both. A Tag that supports BAP PIE shall support being inventoried in parallel with passive PIE Tags without special actions required on the part of the Interrogator. It may also support being inventoried in special battery only Query rounds by use of the optional *Flex_Query* command. The *Flex_Query* command combines the function of a *Select* command and *Query* command, which allows taking targeted types of Tags into Query rounds with a single command.

For BAP PIE Tags the beginning of passive persistence **inventoried** flag and state machine state timeout are controlled by an on-Tag timer function of some type, and not dictated by loss of signal as occurs with passive Tags. This timer function allows the BAP PIE Tag to survive brief signal fades without unnecessarily losing state machine state or **inventoried** flag state, in particular on the S0 flag with its passive persistence definition of "None".

Manchester: The Manchester mode features an improved forward link using Manchester modulation that is capable of taking the performance of BAP Tags to the limits of the physically possible. A Manchester Tag shall also support PIE, though this support requirement is discontinuous as described below in this clause. The Manchester forward mode is orthogonal to the PIE mode in the sense that Tags of each mode do not accidentally decode commands of the other mode, thus providing some degree of interference resistance. The Manchester mode also features a **hibernate** state where a low data rate *Activation* command may bring the Tag up to a fully operational mode.

Interrogator BAP support requirements: Type C Interrogators have no requirements to support any commands or features as described in this clause. Their base requirements are as given in Clause 6. However, if an Interrogator that is specified by its manufacturer as explicitly supporting BAP PIE Tags, then it should also support the *Flex_Query* command of 7.4.1 (see Annex Q for additional recommendations).

If an Interrogator is specified by its manufacturer as explicitly supporting Manchester, then it shall also support the physical modulation form and mandatory commands of 7.5 and its sub-clauses.

BAP Type C Tag PIE support requirements: Type C Interrogator-Talks-First Battery Assisted Passive Tags are required to support PIE. The Tag may satisfy this requirement with either the continuously available PIE of Clause 6, or with the battery supported and optionally duty cycled "Battery Assisted Passive PIE" or BAP PIE of clause 7.4. Though it is not required for BAP PIE to support better sensitivity than passive PIE, it is typical for the battery to allow improved sensitivity. If BAP PIE is duty cycled, the duty cycle maximum **sleep** time shall meet the requirements of 7.4.

The requirement for PIE support is temporarily suspended for Manchester Tags that have been activated in Manchester mode, for programmable Tags that are programmed differently by the user, and for Tags that are authorized to self-adjust their duty cycle in case of a depleted battery state. Also, a Tag that has a user controlled ON-OFF switch or other power-down control method outside of the standard air interface may when powered down be completely unresponsive.

BAP Tags with depleted batteries may be unresponsive, or may function in a purely passive PIE mode as described in Clause 6 where they are completely powered by the Interrogator RF field. Whether or not a BAP Tag will reply with a depleted battery is referred to as the Dead Battery Response (DBR) capability.

BAP Tags have the option to recalculate their storedCRC (see 6.3.2.1.2.1) upon entering the **battery ready** state. Alternatively, they may use a manufacturer specific implementation that meets the specifically BAP Tag modification to 6.3.2.1.2.1 to have the correct storedCRC at the time that the Tag may receive a *Query* command that brings it into a query round. For example, in a BAP Tag with continuous power a flag in the Tag may tell its controller that this update has been made and there is no current need to recalculate the storedCRC.

BAP Tags may but are not required to support Extended Protocol Control (XPC).

BAP Tags may but are not required to support any of the optional commands in Clause 6.

BAP Tag physics and degree of power consumption allows for Tag sensitivity that ranges from approximately -10 dBm to -60 dBm without an RF low noise amplifier, thus leading to wide range of performance and interference control requirements (see Annex Q and Annex R for more detail). This performance may vary with battery state or particular Tag programming.

The term "Low Power Receive" or "LPR" is defined to mean a Tag that features a BAP PIE receive mode that is lower power and sensitivity than an additional receive mode that Tag also possesses. For example, when in LPR mode a Tag might not make use of an optional on-Tag amplifier that provides better sensitivity but consumes additional power when it is on. LPR is not the same as Dead Battery Response, as DBR implies that the Tag fully operates from energy harvested from the RF field. LPR only means that the BAP Tag has an ultra-low power mode in addition to a higher power consumption mode that may be valuable in operational planning.

7.3 Battery Assisted Passive inventoried flag and state machine behaviour modifications

7.3.1 Modification to ready state and power-down support for BAP Tags

All Interrogator-Talks-First Type C BAP Tags shall support a variation of the passive **ready** state referred to as the **"battery ready"** state. The **battery ready** state is logically identical to the passive **ready** state with these two exceptions:

1) The **battery ready** state is battery supported and does not require the presence of an RF field.

2) In the case of BAP PIE Tags that do not support Battery Saver Mode, the **battery ready** state supports **inventoried** and **selected** flag persistence timer operation (see sub-clause 7.4.2) instead of these operations being performed in the implied deenergized state as is done for passive Tags.

Manchester Tags conduct their own form of persistence timing in the "**stateful hibernate**" state (see Figure 7.35). This is not the "inventory state holding" persistence of clause 6.3.2.2 and Table 6.20. Instead it is a persistence to reject new activations aimed at Tags that have not been recently inventoried. If the Tag was not successfully inventoried before an INACT_T or Global Timeout took the Tag back to the **hibernate** state, then the Tag shall be receptive to activations aimed at recently uninventoried Tags.

In BAP PIE Tags that support a Battery Saver Mode the persistence timing operations occur in similar "stateful" variations of the other states in the "**sleep-listen** mode".

The Tag types that support various "low power modes", meaning BAP PIE with Battery Saver Mode and Manchester with its support of hibernation, are exempted from the timing requirements of sub-clause 6.3.1.3.4 when in their associated low power mode. Instead they have more relaxed but still constrained timing limits, such as are described in sub-clause 7.4.2 for BAP PIE. BAP PIE Tags that do not implement Battery Saver Mode will usually be in the **battery ready** state when entering the field of an Interrogator, and in that state they are subject to sub-clause 6.3.1.3.4. All Tags in the **battery ready** state shall support the timing requirement of sub-clause 6.3.1.3.4.

It is highly desirable for battery Tags to have a means of avoiding a permanent powered condition which would rapidly deplete their batteries. Necessary support for this in the case of BAP PIE Tags is provided in sub-clause 7.4.2 on Battery Saver Mode. Tags supporting any of the low power modes (**hibernate** or **sleep-listen**) shall support at least one of the "return to low power" timers INACT_T and Global Timeout (see sub-clause 7.3.2).

7.3.2 Signal loss tolerance via timer (mandatory)

7.3.2.1 Signal loss immunity requirements

Received signal strength as viewed at the Tag suffers strong time variation due to multipath fade, resulting in rapid signal loss and return of signal. The time period of these signal nulls under typical

operating conditions is in the range of approximately 1 to 100 ms (see Annex Q for detailed information) and suffers attenuations of up to tens of dB. Tags may also temporarily completely lose Interrogator signal due to deliberate Interrogator actions such as frequency hopping.

The passive Tag generally loses its operating power within a few forward symbol periods of loss of signal. It then must lose its state machine position (go to the implied de-energized state). At this time passive Tags begin their inventory persistence (a timing function controlling flag state) operation so that they temporarily retain **inventoried** state.

BAP Tags do not lose power until the battery is depleted, hence this forced passive Tag behaviour does not apply. The BAP Tag may take appropriate advantage of its battery by showing a general behaviour whereby the Tag holds its operating state (both **inventoried** flag and state machine state) over temporary loss of signal relative to its sensitivity level. A still more sophisticated behaviour the Tag may display is to hold its state for an interval of time from the last valid command received (environmental validation), so that the Tag recognizes that signals above threshold may actually be from interfering sources that the Tag should decline to allow holding the Tag active. It should be noted that these two behaviours have an opposite behaviour with regards to when timers are in operation. When a timer is used to time loss of signal relative to a threshold, the timer begins when signal is lost and so is normally not running. When a timer is used to measure time from last valid command or preamble, the timer begins upon entering the **battery ready** state, and is reset to begin again on every valid command or preamble, so the timer is always running. Note also that a timer function that validates the environment can avoid the Tag being trapped in an on state in the presence of a strong interferer.

Tag reaction to signal loss is controlled by one or both of the below specified timers, which are applicable to all the BAP Tag types and which may have more advanced behaviours of command recognition. These are the INACT_T and Global Timeout timer functions. The INACT_T function is the primary timer function that holds the Tag in the state it is in for some period of loss of signal or absence of valid commands. Some form of INACT_T function is mandatory, though it may be specified as a reaction time that applies without requiring an explicit timer. The Global Timeout is an optional timer that can recover the Tag from being trapped on in the presence of an interfering carrier. It is thus most useful for situations where environmental validation is not performed, since a single timer with environmental validation can suffice to both survive fades and not be trapped on by interference. These timers are described in detail in the following sub-clauses.

7.3.2.2 INACT_T timer

The period of time the PIE or Manchester BAP Tag maintains its state within the singulation and access state machine and by which it delays the initiation of flag persistence timers without received RF signal (or optionally without correctly decoded signals) is governed by a timer whose timeout period is “INACT_T”. This timer is defined for both BAP PIE and Manchester Tags. Values for INACT_T range from near zero (microseconds) to a few seconds, with 100 ms being a typical value to survive multipath fades typical in the modest velocities of RFID systems. INACT_T also allows for interrogation rounds to span over multiple frequency hopping intervals as used in countries where frequency hopping is a regulatory requirement.

In general there are two modes of INACT_T timer operation, specified as follows:

1. **Threshold INACT_T.** The Threshold INACT_T form begins timer operation upon loss of signal relative to the manufacturer determined threshold. After the manufacturer specified INACT_T period the Tag returns to the **stateful sleep** or **stateful low power listen** state for BAP PIE Tags featuring Battery Saver Mode, or to the **hibernate** state for Manchester (see Figure 7.35). With this simple form of INACT_T behaviour there is the chance that an interfering source of power above the threshold can cause a Tag to remain in a higher power mode. This is referred to as an “interference trap”. For BAP PIE Tags using Threshold INACT_T the manufacturer has the option of providing a separate “Global Timeout” timer (see 7.3.2.3) that will defeat the interference trap and return the Tag to the **battery ready** state. For Manchester Tags using Threshold INACT_T the inclusion of a

Global Timeout is mandatory, and it shall return Manchester Tags to **hibernate** in the presence of interference.

If a BAP Tag implements Threshold INACT_T, its timing actions need not be with respect to a single RF signal level. Hysteresis may be employed where the Tag starts the timer running (loss of signal) on one signal level and resets the timer (reacquisition of signal) on a different signal level. It is recommended to implement hysteresis between an "RF signal detected" level and an "RF signal absent" level to avoid bounce at the threshold level.

If a BAP Tag implements Threshold INACT_T, then the monitoring of signal need not be continuous. The Tag may sample signal strength at discrete times, and internally consider signal to be present or absent after a number of such samples are taken and weighed according to an algorithm of the manufacturer's choice. The total time period applicable to INACT_T then becomes the sum of the individual periods of the number of samples taken to reach this decision.

Some INACT_T behaviour is specified as mandatory even if the Tag does not have actual timers, as there is always some greater than zero time delay before the Tag reacts to signal loss. If the Tag does not possess an actual INACT_T timer, then there is a small period of time generally in the range of microseconds before the Tag reacts to signal loss, and this time shall be specified by the Tag manufacturer as a fixed Threshold INACT_T period. When INACT_T is not supported with a timer, Tags will eventually reset from an inventory or access state to a low power mode via a failsafe "Global Timeout" timer (see 7.3.2.3).

If INACT_T is not supported with a timer then it shall be considered as INACT_T = 0 (or near zero as specified by the Tag manufacturer) for purposes of persistence (see Table 7.1). The no explicit timer case shall be considered as INACT_T = Infinity for purposes of how long a Tag holds a state upon loss of signal or loss of valid Type C preambles or commands, with the Global Timeout function then recovering from a BAP Tag being trapped in an ON state.

2. **Validated INACT_T.** This INACT_T form starts timer operation upon the Tag entering the **battery ready** state, and resets the timer to zero to begin again upon reception of a valid command or preamble. In other words, this INACT_T form validates the environment before resetting the INACT_T timer. If the INACT_T timer expires without a preamble or command causing reset, then the Tag returns to either the **stateful sleep** or **stateful low power listen** state for BAP PIE, or to the **hibernate** state for Manchester (see Figure 7.35). It is left to manufacturer's options what it chooses to constitute a valid command or how these may vary as a function of Tag state. For example, if a Manchester Tag is awakened from **hibernate** with Session Locking (see **Table 7.10** and its notes) in effect, then the may consider that only Interrogator commands with a **inventoried** flag match constitute valid commands for resetting the Validated INACT_T timer.

   BAP PIE Tags that support Battery Saver Mode shall implement at least one of INACT_T or Global Timeout. Further details of the behaviour of BAP PIE Tags upon expiration of the INACT_T timer are described in sub-clause 7.4.2. Since Global Timeout is specifically defined to apply to return to a low power mode, BAP PIE Tags that do not support Battery Saver Mode shall support INACT_T to provide a means to force them back to the **battery ready** state upon a sufficient period of loss of signal or loss of Type C commands. The expiration of the INACT_T or Global Timeout functions may override the responses to T2 timeout described in 6.3.1.6.

   A BAP Tag that is programmed to INACT_T greater than 500 ms shall apply a maximum of 500 ms delay to the start of its inventory persistence timers when operating in BAP PIE mode (see Table 7.1).

   The INACT_T timer shall display an accuracy of +/-40% accuracy over nominal temperature range of -25 to + 40 °C, and (if supported) a +/-50% accuracy over extended temperature range of -40 to +65 °C.

7.3.2.3 Global Timeout timer and Selective Global Timeout

This optional timer causes the Tag to fall back to a low power state (**sleep**, **low power listen**, or **hibernate** as appropriate) a relatively long time after it was activated to the Normal Mode. The Global Timeout may at the option of the Tag manufacturer be of either of these two forms:

1. Global Timeout. This form forces the Tag back to its low power mode regardless of Tag state, such as the Tag currently being involved in a legitimate inventory round. If implemented in this way, then in order to reduce the odds of an intended inventory round from being interrupted, this Global Timeout shall be 4 seconds or more. Global Timeout begins running when the Tag enters the **battery ready** state, and upon expiration and regardless of what the Tag is doing takes the Tag back to the **stateful sleep** or **stateful low power listen** state for BAP PIE, or to the **hibernate** state for Manchester (see Figure 7.35). It may thus affect a Tag engaged in a legitimate inventory operation.

2. Selective Global Timeout. This form is selective as to Tag state. Selective Global Timeout has the behaviour that it begins running when the Tag enters the **battery ready** state, but that the timer refreshes upon the Tag detecting legitimate commands. This generally avoids causing a Tag engaged legitimate operations from timing out. An example of selective behaviour is resetting Selective Global Timeout upon receiving valid commands, with different options possible as to what is considered a valid command. These options are at the choice of the Tag manufacturer, but should be described in product data sheets. The Selective Global Timeout, if implemented, shall have a minimum duration of at least 50 ms.

The Global Timeout shall display an accuracy of +/-40% accuracy over nominal temperature range of -25 to +40 °C, and (if supported) +/-50% accuracy over extended temperature range of -40 to +65 °C.

BAP Tags shall support at least one of the INACT_T or Global Timeout timers. The only currently specified requirements as to choices are that:

1. BAP PIE Tags that do not implement Battery Saver Mode shall implement INACT_T (such Tags do not have a low power state for Global Timeout to apply to).

2. For Manchester Tags the INACT_T timer is optional if Global Timeout is provided, but if INACT_T is provided then it shall be of the Validated INACT_T form, and the minimum time of INACT_T shall be 50 ms. Note that Validated INACT_T allows a single timer to avoid the interference trap problem, and so with Validated INACT_T a Global Timeout is usually not necessary (an exception could be for Tags that spend long periods of time near active Interrogators). More detailed behaviour of Manchester Tags is described in sub-clause 7.5.3.6.

7.3.3 Modified persistence of BAP PIE inventory flags (optional)

BAP PIE Tags shall provide for persistence behaviour either identical to that of passive Tags or enhanced with immunity to fade via the INACT_T timer and more precise persistence maximum values. Manchester Tags shall support persistence in a different way, via returning to the **hibernate** state and running an accurate persistence timer that (unlike passive or BAP PIE) has identical behaviour for all four **inventoried** flags.

For BAP PIE Tags the optional specification on maximum flag persistence can reduce the incidence of needing to inventory Tags from **inventoried** flag state *A*→*B* and then again from *B*→*A*. The specification fits within and is backward compatible to the persistences defined for pure passive in Clause 6 in Table 6.20. Tags that have Dead Battery Response may in the case of a dead battery continue to support the persistences of Table 7.1, or may revert to meeting the more relaxed requirements of of Clause 6.

The values for optional BAP PIE flag persistence are as shown in Table 7.1. The flags in this table shall have the same functions as the passive **inventoried** and **selected** flags, simply with the specified maximums and delay time INACT_T that may be chosen to apply while the Tag has battery power. If a BAP PIE Tag that supports Battery Saver Mode does not support INACT_T, then for the purposes of Table 7.1, INACT_T shall be considered as zero.

Table 7.1 — Optional improved Battery Assisted Passive PIE flag persistence values (the Tag manufacturer may comply with the persistence values of this table, but is only required to comply to the more relaxed passive persistence values of Table 6.20)

Flag	Persistence (See NOTES 1-4)
S0 **inventoried** flag	Above sensitivity threshold or successful decoding: Indefinite Below sensitivity threshold or failure to decode : Zero (following INACT_T, see NOTE 1)
S1 **inventoried** flag[a]	Above sensitivity threshold or successfully decoding: –25 °C to +40 °C: 500 ms < persistence < 5 s –40 °C to -25 °C and +40 °C to +65 °C: Not specified if Tag does not cover extended temp range, but 500 ms < persistence < 5 s if Tag does cover extended temp range Below sensitivity threshold or failure to decode: –25 °C to +40 °C: 500 ms < persistence < 5 s –40 °C to -25 °C and +40 °C to +65 °C: Not specified if Tag does not cover extended temp range, but 500 ms < persistence < 5 s if Tag does cover extended temp range
S2 **inventoried** flag[a]	Above sensitivity threshold or successfully decoding: Indefinite Below sensitivity threshold or failure to decode: –25 °C to +40 °C: 2 s < persistence < 20 s (beginning after INACT_T) –40 °C to -25 °C and +40 °C to +65 °C: Not specified if Tag does not cover extended temp range, but 2 s < persistence < 20 s if Tag does cover extended temp range (beginning after INACT_T)
S3 **inventoried** flag[a]	Above sensitivity threshold or successfully decoding: Indefinite Below sensitivity threshold for failing to decode: –25 °C to +40 °C: 2 s < persistence < 20 s (beginning after INACT_T) –40 °C to -25 °C and +40 °C to +65 °C: Not specified if Tag does not cover extended temp range, but 2 s < persistence < 20 s if Tag does cover extended temp range (beginning after INACT_T)
selected (**SL**) flag[a]	Above sensitivity threshold or successfully decoding: Indefinite Below sensitivity threshold or failing to decode: –25 °C to +40 °C: 2 s < persistence< 20 s (beginning after INACT_T) –40 °C to -25 °C and +40 °C to +65 °C: Not specified if Tag does not cover extended temp range, but 2 s < persistence < 20 s if Tag does cover extended temp range (beginning after INACT_T)
Optional **C** flag	Above sensitivity threshold or successfully decoding: Indefinite Below sensitivity threshold or failing to decode: –25 °C to +40 °C: 0 s ≤ persistence< 5 s (beginning after INACT_T) –40 °C to -25 °C and +40 °C to +65 °C: Not specified if Tag does not cover extended temp range, but 0 s ≤ persistence < 5 s if Tag does cover extended temp range (beginning after INACT_T)

Table 7.1 *(continued)*

Flag	Persistence (See NOTES 1-4)
Optional security timeout	Above sensitivity threshold or successfully decoding: –25 °C to +40 °C: 20 ms ≤ persistence< 200 ms –40 °C to -25 °C and +40 °C to +65 °C: Not specified if Tag does not cover extended temp range, but 20 ms ≤ persistence < 200 ms if Tag does cover extended temp range Below sensitivity threshold or failing to decode: –25 °C to +40 °C: 20 ms ≤ persistence< 200 ms –40 °C to -25 °C and +40 °C to +65 °C: Not specified if Tag does not cover extended temp range, but 20 ms ≤ persistence < 200 ms if Tag does cover extended temp range

NOTE 1 The beginnings of the persistence time periods for S0, S1, S2, S3, and **SL** are all delayed by manufacturer specified signal loss or failure to decode tolerance time INACT_T. For the purposes of this table, if INACT_T is not supported then it shall be regarded as zero (or near zero hardware delay as specified by the Tag manufacturer), in which case the persistence timing begins just as it does for passive Tags.

NOTE 2 A BAP Tag that is programmed to INACT_T greater than 500 ms and complying with this table shall apply a maximum of 500 ms delay to the start of its **inventoried** persistence timers when operating in BAP PIE mode.

NOTE 3 This standard uses *A* and *B* as proxy logic states for **inventoried** flags with the understanding that the electrical logic state is implementation dependent. Similarly the **selected** flag uses proxy states **SL** (asserted) and **~SL** (deasserted). The default state (following persistence) of the **inventoried** flags is state *A*, and that of the **selected** flag is **~SL**. Thus inventory timers controlling persistence need only operate if the state is *B* or SL, and upon expiration of the timer these flags change state from *B*→*A* and from **SL** to **~SL**.

NOTE 4 The accuracy of these flags is required to at least match the passive Tag case, which is as follows. Those flags denoted with a superscript "a" in Table 7.1 shall for a randomly chosen and sufficiently large Tag population shall behave such that at least 95% of the Tag persistence times shall meet the persistence requirements in Table 6.20, with at least a 90% confidence interval. Tags that have accurately specified tolerances on persistence timers may meet these requirements by reducing their range of persistence to allow for timer error. See Q.10 for analysis.

7.4 Battery Assisted Passive PIE (optional)

7.4.1 Flex_Query command (optional)

The basic function of this optional command is to provide a "Tag Type Select" function that allows for selecting the categories of Tags to be included in the query round without requiring a separate *Select* command. This selection allows for the Interrogator to bring nearly any combination of Tag types into Query rounds.

Tags that support Dead Battery Response and *Flex_Query* may continue to support *Flex_Query* with a depleted battery.

Table 7.2 — *Flex_Query* command (optional) (See NOTES 1-3)

	Command	Tag Type Select	SS Resp NOTE 2	MIIM Resp	DR	M NOTE 3	TRext	Sel	Session	Tar get	Q	CRC-5
# of bits	8	12	1	1	1	4	1	2	2	1	4	5
descrip-tion	11001111	NOTE 1	0: No 1: Yes	0: Disable 1: Enable As defined in ISO/IEC 29143	0: DR=8 1: DR=64/3	0000: M=1 0001: M=2 0010: M=4 0011: M=8 0100: M=16 0101: M=32 0110: M=64 0111 to 1111: RFU	0: No pilot tone 1: Use pilot tone	00: All 01: All 10: ~**SL** 11: **SL**	00: S0 01: S1 10: S2 11: S3	0: *A* 1: *B*	0–15	

NOTE 1 See Table 7.3 for full definition of Tag types to be pulled into the Query round. Possible implementation of *Flex_Query* by passive Tags is a future option.

NOTE 2 The Simple Sensor Response flag authorizes automatic transmission of Simple Sensor data in cases where sensor Tags have been selected. Tags without Simple Sensors shall ignore the SS Response flag.

NOTE 3 M values of 16, 32, and 64 are optional for Tags implementing the *Flex_Query* command. A Tag that receives a *Flex_Query* command for an unsupported M value shall ignore the command.

Table 7.3 — *Flex_Query* command Tag Type Select field (see NOTES 1-2)

Interpretation NOTE 2	RFU	MIIM	Sensor Alarm	Full Function Sensor	Simple Sensor	RFU	RFU	RFU	Battery Assisted Passive	RFU	Passive NOTE 1 (RFU)
1	1	1	1	1	1	1	1	1	1	1	1
0: Inclusive 1: Exclusive NOTE 2		0: Disable 1: Enable As defined in ISO/IEC 29143	0: Disable 1: Enable	0: Disable 1: Enable	0: Disable 1: Enable				0: Disable 1: Enable		0: Disable 1: Enable

NOTE 1 *Flex_Query* is not currently defined for passive Tags, but the Passive select field is noted for when it becomes defined.

NOTE 2 This field switches the interpretation of the following selection fields between “inclusive” (the Tag responds if it matches any selection) and “exclusive” (the Tag responds only if it matches all selected criteria).

BAP PIE Tags that implement *Flex_Query* shall respond to *Flex_Query* by beginning a new Query round from the **battery ready**, **arbitrate**, **reply**, **acknowledged**, **open**, and **secured** states.

Just as in the normal *Query* command, the *Flex_Query* command is preceded by R=>T Preamble and not R=>T Frame-Sync.

There is no reply to *Flex_Query* unless the Tag rolls zero on its slot counter, in which case it shall reply with its RN16 as in Table 7.4.

Table 7.4 — Tag reply to a *Flex_Query* command (only if slot counter = 0)

	Response
# of bits	16
description	RN16

7.4.2 BAP PIE detailed operation including optional Battery Saver Mode

This sub-clause provides detailed specification of BAP PIE mode both with and without Battery Saver functionality. A state machine description is used to provide a logical framework applicable to both of these cases and which integrates with the passive Type C state diagram of Figure 6.21. This passive state diagram is a subset of the BAP PIE state diagram. Whether the Tag formally follows a state machine according to Figure 7.27 is an implementation issue, but from the Interrogator point of view the Tag implementing BAP PIE shall act as if it does according to the functional behaviours and times specified in this sub-clause.

A Tag providing BAP PIE mode may employ Battery Saver functionality to reduce the average battery current drain when the Tag is not in the field of an Interrogator. This mode has two main options of which a Tag using this mode shall provide at least one and may provide both.

The first option is the use of Low Duty Cycle (LDC) mode. LDC is defined as the following ratio:

"**listen** state" time / cycle time, where cycle time = "**listen** state" time + "**sleep** state" time.

In **listen** state the Tag samples the presence of Interrogator RF radiation, generally with higher sensitivity and relatively high current drain. In **sleep** state all Tag circuits are disabled, except the duty cycle timer, draining very low current from the battery.

Neglecting small changes due to processing within Query rounds, the average battery current drain is approximated by:

sleep state current * (1- LDC) + (**listen** state current) * LDC

LDC causes a delay in the Tag "Power-up" time defined in clause 6.3.1.3.4. The maximum power-up delay of the BAP PIE Tag in Low Duty Cycle Battery Saver Mode is given by the sum of the cycle time and the appropriate delay as described in clause 6.3.1.3.4.

The second option is a low power mode with a continuous built in listen capability (typically not as sensitive as a part time **listen** mode). If the Tag supports this continuous low power listen capability as its lowest power state, then this state is referred to as "**low power listen**" instead of "**sleep**". See Figure 7.35 for a suggested state machine diagram.

The Tag has relaxed timing requirements before being required to be in the **battery ready** state following initiation of Interrogator RF carrier while using Battery Saver Mode. This relaxed time to **battery ready** allows for these more sophisticated Tags to provide for settling time of regulators and clock sources that may be off while in the **sleep-listen** mode (see below in this sub-clause),

The possible BAP PIE modes are specified in detail as follows and with reference to Figure 7.35. This figure provides a complete state description of BAP PIE Tags that both support or do not support a Battery Saver Mode. If a Battery Saver Mode is supported, this figure also describes both duty cycled and non-duty cycled behaviour. The figure also covers use of timers for INACT_T and Global Timeout.

The states in Figure 7.35 are defined as follows:

sleep: A near zero power mode where the Tag is not responsive to RF signals. The Tag transitions from **sleep** to **listen** only under control of an internal timer, and does so within a manufacturer selected **sleep** to **listen** time S_to_L. It is in **sleep** for the manufacturer chosen period **sleep** time or ST. ST may also be a range of times that may change under operating conditions, such as to avoid interference induced false wake-ups.

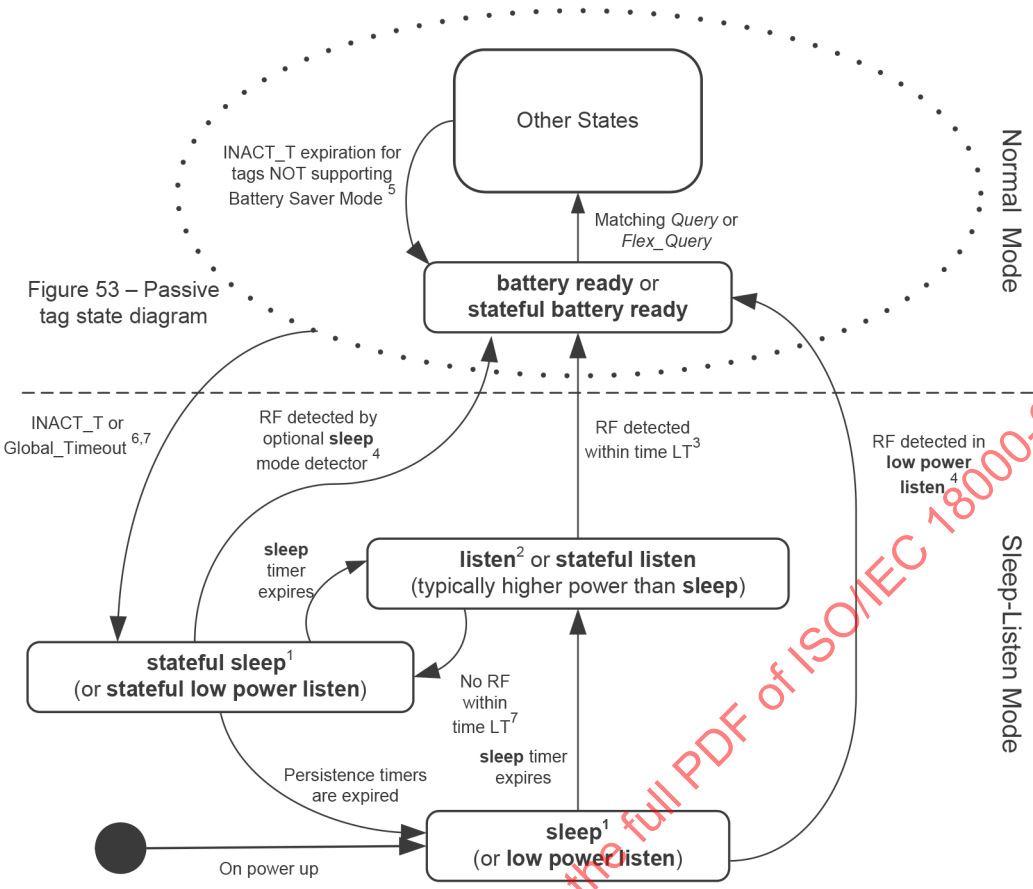
stateful sleep: This is the **sleep** state with the addition of one or more **inventoried** or **selected** flag persistence timers in operation. The Tag may also cycle through the **stateful sleep** state while briefly checking if such timers are in operation and if not then transition to **sleep**. It is comparable to the **stateful hibernate** modes of Manchester. It responds to internal duty cycle timer control as does **sleep**.

low power listen: A replacement for **sleep** where the Tag is continuously responsive to RF signals, though generally using low power and thus only achieving modest sensitivity. It is typically used without duty cycling to a more sensitive state, but may be used in combination with duty cycling. The Tag shall transition out of **low power listen** to **battery ready** upon an RF signal greater than its manufacturer defined threshold within manufacturer selected **low power listen** to **battery ready** time LPL_to_R of 20 ms or less. The Tag may optionally also transition from **low power listen** to **listen** under control of an internal timer, and if so then the Tag does this within **low power listen** to **listen** time LPL_to_L.

stateful low power listen: This is the **low power listen** state with the addition of one or more **inventoried** or **selected** flag persistence timers either in operation or being checked for operation. It is also comparable to the **stateful hibernate** mode of Manchester. It responds to RF signals as does **low power listen**. The Tag is considered to transition from **stateful low power listen** to **listen** upon confirming expiration of the last of its **inventoried** or **selected** flag persistence timers.

listen: A part time state where the Tag is responsive to RF signals, typically but not necessarily with improved sensitivity. This state is used in a Duty Cycled Battery Saver Mode. The Tag is not required to decode signals in the **listen** state, but shall at least detect their presence. The Tag shall transition from **listen** to **battery ready** upon either receiving a signal above a manufacturer defined threshold or if validating the environment upon correctly decoding either a PIE preamble or command (the degree of environmental validation is up to the manufacturer). The transition from **listen** to **battery ready** shall occur within manufacturer selected **listen** to **battery ready** time L_to_R of 20 ms or less. The Tag shall remain in **listen** for manufacturer selected **listen** time LT, and if no RF is detected or no correct PIE symbol or command is decoded the Tag shall then return to **sleep**, **stateful sleep**, **low power listen**, or **stateful low power listen** as appropriate. Transitions to these low power states shall be within manufacturer chosen **listen** to **sleep** time L_to_S.

stateful listen: Identical to **listen** except at least one persistence timer is in operation. Upon expiration of the persistence timers the **stateful listen** state is considered to transition to the **listen** state.



Notes:

1. **Sleep** may optionally also "listen", though normally at reduced sensitivity, in which case the name of the state changes to **low power listen**.
2. If **sleep** has its own "listen", then the separate **listen** state (generally at improved sensitivity) is optional. The listen timer period may be increased and/or **sleep** period decreased by the Tag in response to recent activity. They may also be adjusted by the tag near the end of battery charge in order to increase battery life.
3. A transition from **listen** or **stateful listen** to **battery ready** occurs within L_to_R time <= 20 ms.
4. A transition from **low power listen** or stateful low power listen to **battery ready** (if supported) occurs within LPL_to_R time <=20 ms.
5. If the BAP Simple PIE tag does NOT support Battery Saver Mode, then it runs its persistence timers (holds inventory state) in the **battery ready** state.
6. If the BAP Simple PIE tag does support Battery Saver Mode, then it runs its persistence timers (holds inventory state) in the **stateful sleep** or **stateful low power listen** state, and upon expiration transitions to the full **sleep** state or **low power listen** state. These timers only operate if the inventory state is B or the **selected** state is **SL**, and upon expiration the states change to A or **~SL** as appropriate.
7. The "stateful" states mean timers are running or that the tag is checking if they are running. The tag manufacturer has the option of going through the stateful states to check if timers are running, or checking in the prior state and going straight to sleep or **low power listen**.

Figure 7.27 — BAP PIE and Battery Saver Mode state diagram

Battery ready is specified in sub-clause 7.3.1. The remainder of the states under the block called "Other States" are the passive Tag states other than **ready** shown in Figure 6.21. Being in the **battery ready** or any of the passive Tag states is referred to as "Normal Mode". Being in any of **sleep**, **stateful sleep**, **low power listen**, **stateful low power listen**, or **listen**, is referred to as being in "Sleep-Listen Mode".

If a BAP PIE Tag transitions from any stateful state to the Normal Mode, it shall reset its persistence timers to zero and halt their operation while holding its current flag states indefinitely (while RF or decoded RF has not been absent longer than INACT_T) with the exception of the S1 **inventoried** flag.

The transitions between states in the Normal Mode are identical to that exhibited for passive Tags with the exception of delays and return to Sleep-Listen Mode as dictated by the manufacturer's implementation of INACT_T. INACT_T may be implemented with respect to RF signal strength in relation to a threshold, or with relation to correct receipt of valid Type C preambles or commands.

INACT_T normally delays the start of persistence timeout and state machine state changes upon loss of RF or decoded RF. However, according to sub-clause 6.3.1.6 on Link Timing, the maximum time of the T_2 timer and its specified state changes apply instead of INACT_T while the Tag is in the **reply** or **acknowledged** states.

Global Timeout does not take RF signalling or Tag position in the Normal Mode state machine into account, but Selective Global Timeout may take these factors into account. See sub-clause 7.3.2.3.

Environmental validation for BAP PIE: If INACT_T is supported relative to valid Type C preambles or commands (Validation INACT_T), this allows the opportunity to validate the radio environment that awakened a Tag as a legitimate Type C environment. Validation INACT_T shall be conducted as follows: The Tag transitioning from a Sleep-Listen Mode to **battery ready** shall in **battery ready** listen for valid Type C preambles or commands at the manufacturer's choice. If the Tag does not decode valid Type C signals within INACT_T, it shall return to the **sleep** or **low power listen** state as appropriate. The Tag may upon manufacturer choice increase its **sleep** time ST to avoid non-Type C signals causing further false wake ups. If it does detect valid Type C signals within INACT_T, the Tag shall reset the INACT_T timer. The Tag may then perform one of two actions:

1. Continue to reset the INACT_T timer indefinitely upon receipt of valid preambles or commands. In this case Global Timeout would be recommended to eventually force the Tag back to the **sleep-listen** mode.

2. Wait for valid *Select* and *Query* commands (or *Flex_Query* command) to take it into a Query round, while resetting the INACT_T timer for a limited number of manufacturer selected times. If the Tag has not entered a Query round upon expiration of INACT_T for the manufacturer selected number of times, it may return to the Sleep-Listen Mode. If the Tag is taken into a Query round it shall again reset the INACT_T timer.

While a Tag is in a Query round it shall upon receipt of further valid Type C preambles or commands continue to reset the INACT_T timer. See Annex Q for description of various applicable command validation methods. If signal is lost the Tag shall hold state machine state and not begin persistence timer operation (if the Tag supports BAP PIE persistence as defined in Table 7.1) until the expiration of the INACT_T timer (with the exception of the requirements of sub-clause 6.3.1.6). Tags that support a Battery Saver Mode shall then transition to the **sleep**, **stateful sleep**, **low power listen**, or **stateful low power listen** state as appropriate (the stateful states are used if any **inventoried** flag is in state *B* or if **selected** is in state **SL**). Tags that do not support Battery Saver Mode shall upon expiration of INACT_T transition to the **battery ready** state and (if the Tag supports BAP PIE persistence as defined in Table 7.1) begin persistence timer operation if any **inventoried** flags are in state *B* or if **selected** is in state **SL**. That condition may be considered as a **stateful battery ready** state, and upon expiration of persistence timers the **inventoried** and **SL** flag state will change as appropriate.

Environment not validated: In this case INACT_T is not supported or if supported is responsive to signal strength without regard to decoding of valid Type C signals (Threshold INACT_T). The Tag transitioning from a Sleep-Listen Mode state to the Normal Mode that supports Threshold INACT_T shall in any Normal Mode state listen for RF signals above its manufacturer defined threshold. When such are detected the INACT_T timer is reset (held in reset if RF signal is continuously detected above threshold), and when signal drops below threshold (optionally a different threshold than that necessary to reset the INACT_T timer) then the INACT_T timer will begin. If the INACT_T timer expires the Tag returns to the Sleep-Listen Mode (if a Battery Saver Mode is supported) or to the **battery ready** state (if a Battery Saver Mode is not supported) as appropriate.

Now the various BAP PIE cases may be completely specified as follows.

Battery Saver Mode not supported: In this case the BAP PIE Tag is in the **battery ready** state when not in the field of an Interrogator, and behaves nearly identically to a passive Tag (though typically with better sensitivity). The Tag responds to the application of RF carrier within the timing limits of sub-clause 6.3.1.3.4 (1.5 ms to be ready to detect a preamble). Since the Tag does not automatically reset to the **battery ready** state upon loss of power, the Tag shall implement INACT_T to force the Tag back to the **battery ready** or **stateful battery ready** (when persistence timer functions are in operation) state when it is not in range of an Interrogator. The Tag manufacturer may support the persistence

requirements of Table 7.1 for BAP PIE Tags (persistence delayed by INACT_T and with the specified maximums), or may support the more relaxed persistence requirements of passive Tags as given in Table 6.20. The Global Timeout is not applicable to this case.

Battery Saver Mode supported via low power listen only: In this case the Tag is not supporting duty cycling of **listen** and maintains a continuous low power **listen** mode. It shall transition to the **battery ready** state upon detecting RF field above its manufacturer determined threshold within LPL_to_R time of 20 ms. This longer allowed time to be in the **battery ready** state is what distinguishes this case from Battery Saver Mode not supported. Since the Tag does not automatically reset to the **low power listen** state upon loss of power, the Tag shall implement one of INACT_T or Global Timeout to force the Tag back to the **low power listen** state when it is not in range of an Interrogator. When the Tag departs Normal Mode it shall transition to either the **stateful low power listen** state (if no timers running is not confirmed before leaving Normal Mode) or **low power listen** state (if no timers running is confirmed before departing Normal Mode). The reason for allowing the Tag the option of briefly transitioning to **stateful low power listen** for the timer check operation is to simplify state machine or firmware operation. The Tag shall transition from the **stateful low power listen** state to the **low power listen** state upon confirming expiration of all persistence timers.

Battery Saver Mode supported via duty cycling only: The Tag shall **listen** (generally in a relatively high sensitivity state) for time LT and **sleep** for time ST. It shall transition to the **battery ready** or **stateful battery ready** state upon detecting RF field while in the **listen** state above its manufacturer determined threshold within L_to_R time of 20 ms. Since the Tag does not automatically reset to the **sleep** state upon loss of power, the Tag shall implement one of INACT_T or Global Timeout to force the Tag back to the **sleep** state when it is not in range of an Interrogator. Upon departing Normal Mode, the Tag shall transition to either the **stateful sleep** (if timers running or to confirm if timers running) or **sleep** state (if it confirms no timers running). The Tag will continue to operate its duty cycle control to transition to the **stateful listen** mode and back while in **stateful sleep**. The Tag shall transition from the **stateful sleep** state to the **sleep** state upon confirming expiration of all persistence timers.

In the **listen** mode the Tag is standing by to enter the **battery ready** state upon application of an RF carrier. The **listen** Time (LT) that the Tag maintains itself in the **listen** state in the absence of RF signal may be selected by the manufacturer. If the Tag does not receive a signal above a manufacturer defined threshold level during the LT interval, then the Tag shall transition back to the **sleep** state. If the Tag does receive any signal above the manufacturer defined sensitivity threshold, then the Tag shall transition to the **battery ready** state and will do so within the timing limit of **listen** to **battery ready** time L_to_R <= 20 ms.

In the **sleep** state the Tag is running a timer of duration **sleep** time (ST) and is waiting to return to the **listen** state upon expiration of the ST timer. ST may be fixed by the manufacturer. ST is typically on the order of 10 ms to 1 s. ST is chosen shorter for higher Tag velocity environments or longer for slow moving environments, with ~100 ms being a typically chosen or programmed value. Though manufacturers are free to select any ST they should specify their chosen ST within their product documentation.

Battery Saver Mode supported via duty cycling and low power listen: In this case the Tag is both supporting duty cycling of **listen** (generally in a higher sensitivity state) and maintaining a continuous low power **listen** mode (generally in a lower sensitivity state) when not in **listen**. It shall transition to the **battery ready** or **stateful battery ready** state upon detecting RF field above its manufacturer determined thresholds within L_to_R or S_to_R time as appropriate. Since the Tag does not automatically reset to the **low power listen** state upon loss of power, the Tag shall implement one of INACT_T or Global Timeout to force the Tag back to the Sleep-Listen Mode when it is not in range of an Interrogator. While in **low power listen** the Tag also operates its **listen-sleep** timer to periodically transition to the **listen** mode to listen in a higher sensitivity state than in the **low power listen** state. Other operation is as described above for duty cycle operation only or low power listen operation only.

Table 7.5 below provides a summary of the various delay times and timers that apply to BAP PIE Tags.

Table 7.5 — Battery Saver Mode timing parameters

Timing parameter	Description	Effects & comments
LT	**listen** time, the time the Tag is listening for any RF above a threshold before the Tag goes back to the **sleep** or **low power listen** state.	Manufacturer defined.
ST	**sleep** time, the low power consumption time the Tag is either not listening or is only listening at low sensitivity.	Manufacturer defined. If Tag only supports a low power continuous "listen" and is not duty cycled to a higher power more sensitive **listen** state, then this time is set to infinity or not applicable.
S_to_L	**sleep** to **listen** time, also used for **low power listen** to **listen** time.	Manufacturer selected, typically << ST but not required to be a particular value.
L_to_R	**listen** to **battery ready** time, a maximum of 20 ms following signal exceeding threshold.	20 ms max, actual value documented in data sheet. Only applicable if Tag supports optional duty cycled **listen** state. This time also applies to **stateful listen** to **battery ready**.
L_to_S	**listen** to **sleep** time.	Manufacturer selected, typically << ST but not required to be a particular value. This time also applies to the **stateful listen** to **stateful sleep**, **listen** to **low power listen**, and **stateful listen** to **stateful low power listen** times.
LPL_to_R	**low power listen** to **battery ready** time, a maximum of 20 ms following signal exceeding threshold.	20 ms max, actual value chosen by manufacturer. Only applicable if Tag supports optional continuous listen while in the **low power listen** state as a replacement for the **sleep** state. This time also applies to the transition from **stateful low power listen** to **battery ready**.
N_to_S	Normal to **sleep** time	Time from any normal state to **sleep**, **stateful sleep**, **low power listen**, or **stateful low power listen**, following expiration of INACT_T or Global Timeout. Normally << than INACT_T or Global Timeout, but not required to be any particular value.
INACT_T	A timer that allows a BAP PIE Tag to temporarily hold its state machine state in the absence of RF signal or in the absence of valid Type C preambles or commands. Is continually reset upon either signal above threshold or decoding of a valid preamble or command. Can apply to Tags that support or do not support Battery Saver Mode.	See for responses for different BAP PIE modes with respect to support of Battery Saver Mode. At least one of INACT_T or Global Timeout shall be supported for BAP PIE and Manchester Tags.
Global Timeout	A timer that allows a BAP PIE Tag to temporarily hold its state machine state in the absence of RF signal or in the absence of valid Type C preambles or commands. Is initiated by Tag transitioning from a **sleep-listen** Mode to the **battery ready** state, and is NOT reset by RF signal strength or correct decoding of preambles or commands. Only applies to Tags that support a Battery Saver Mode.	See for responses for different BAP PIE modes with respect to support of Battery Saver Mode. At least one of INACT_T or Global Timeout shall be supported. At least one of INACT_T or Global Timeout shall be supported for BAP PIE and Manchester Tags.

According to the above table, the total time of the **sleep** and **listen** cycle is ST + S_to_L + LT + L_to_S. This would normally be dominated by the **sleep** and **listen** time intervals.

When a BAP PIE Tag also supports Manchester, that Tag shall maintain any **hibernate** mode timer values and inventory states until it validates the environment as a legitimate PIE environment. That validation shall be to at least properly decode PIE commands. Upon proper validation of environment, the Tag shall clear any **hibernate** mode timers, set all **inventoried** flag states to A, set **selected** flag state to **~SL**, and stand by for Interrogator commands.

7.5 Manchester mode Battery Assisted operation protocol extensions

7.5.1 Introduction

This clause defines optional extensions of the ISO/IEC 18000-63 Type C protocol for Battery Assisted Passive RFID utilizing a Manchester mode forward link. The Manchester mode is optional, but if a Tag or Interrogator uses this optional mode, all requirements in this clause are mandatory. The extensions defined here are intended to maximize Interrogator-to-Tag read ranges and battery life. This is achieved by using DC balanced symbols in the forward link which enable the use of AC coupled inputs to high gain preamplifiers. Tags may be designed with medium to extremely sensitive receivers. For the more sensitive Tags, the Manchester command preamble length provides for sufficient time to switch between two receiver dynamic ranges to cover the large input signal dynamic range. The longer range also requires higher symbol frequency accuracy at the lower data rates which results in higher values of M. Higher accuracy is enabled at lower signal levels in Tags equipped with battery powered oscillators by providing configuration bits in the forward link commands that specify the BLF and that are not subject to measurement accuracies of TRcal and RTcal.

Compatibility with ISO/IEC 18000-63 Type C Interrogators and passive Tags is maximized through the use of methods for channel access similar to ISO/IEC 18000-63 Type C, ensuring effective channel sharing in mixed mode environments. The Interrogator talks first, and the Tags backscatter responses with random slot times identical to ISO/IEC 18000-63 Type C. The Interrogator awakens battery assisted Tags with a specific activation sequence, followed by a selection field which identifies which Tags need to respond. Tags not selected by the *Activation* command quickly return to the lower power search (**hibernate**) mode. The battery assist activation is distinctly different from ISO/IEC 18000-63 Type C commands specified in Clause 6 and thus ISO/IEC 18000-63 Type C passive Tags will not respond, and conversely battery assisted Tags compliant to this clause will not respond to ISO/IEC 18000-63 Type C passive Interrogators in battery assisted mode.

Battery assisted Interrogators currently use the same channel definitions and similar bandwidths as ISO/IEC 18000-63 Type C Interrogators. Therefore, RF coordination and channel sharing concepts apply across both battery assisted and passive modes of ISO/IEC 18000-63 Type C. The UII numbering and formatting, as well as all memory bank definitions and structure, are identical to the ISO/IEC 18000-63 Type C definition.

For battery assisted mode, the higher sensitivities and longer ranges create a higher likelihood of interference between multiple Interrogators and the Tags they are communicating with. The activation mechanism defined in this section provides for the use of Interrogator power level control. Tags can be activated and operated on in sub-groups using lower power levels for Tags that are closer. Tags that have already been singulated or accessed at lower power levels can be silenced for commandable periods while the higher power operations on Tags that are farther away from Interrogators are performed. The use of power levelling is recommended in applications that would allow such system interference reducing techniques.

NOTE As PIE is a required mode and Manchester is an optional mode for Type C, Manchester Tags must support PIE. The definition of "support PIE" is given in clause 7.2.

7.5.2 Physical layer

7.5.2.1 Interrogator-to-Tag (R=>T) communications

7.5.2.1.1 Modulation

Interrogator shall communicate with the Tag using amplitude shift keyed Manchester encoded DSB-ASK, SSB-ASK, or PR-ASK modulation in the forward air-interface link. Tags shall demodulate all three modulation types.

7.5.2.1.2 Data rates

Interrogators shall communicate with Tags using one of the following data rates, while Tags shall support all of the data rates:

8 kbit/s, 16 kbit/s, 32 kbit/s, 64 kbit/s, and 128 kbit/s.

7.5.2.1.3 Frequency accuracy

Interrogator frequency accuracy shall be +/– 10 ppm over the Nominal temperature range of -25 °C to +40 °C, and +/- 12 ppm over the extended temperature range of -40 to +65 °C. These accuracies shall apply for a manufacturer specified time interval following sale of the product. If local regulatory requirements specify tighter accuracy, the Interrogator frequency accuracy shall meet the local regulatory requirements.

7.5.2.1.4 Data encoding

The R=>T link shall use Manchester data encoding, shown in Figure 7.28.

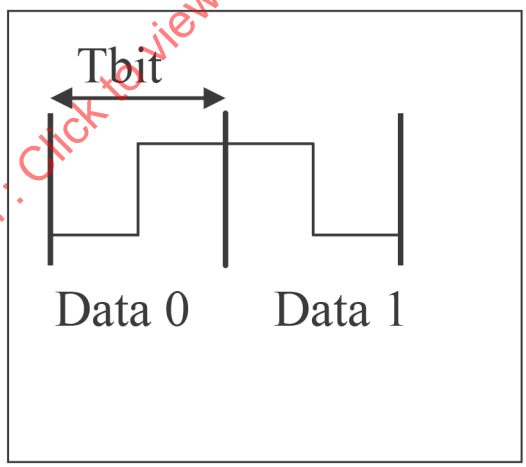


Figure 7.28 — Manchester symbols

7.5.2.1.5 Interrogator to Tag RF Envelope

The RF envelope (in electric field) of the Manchester modulation shall be within the limits specified in Figure 7.29 and Table 7.6.

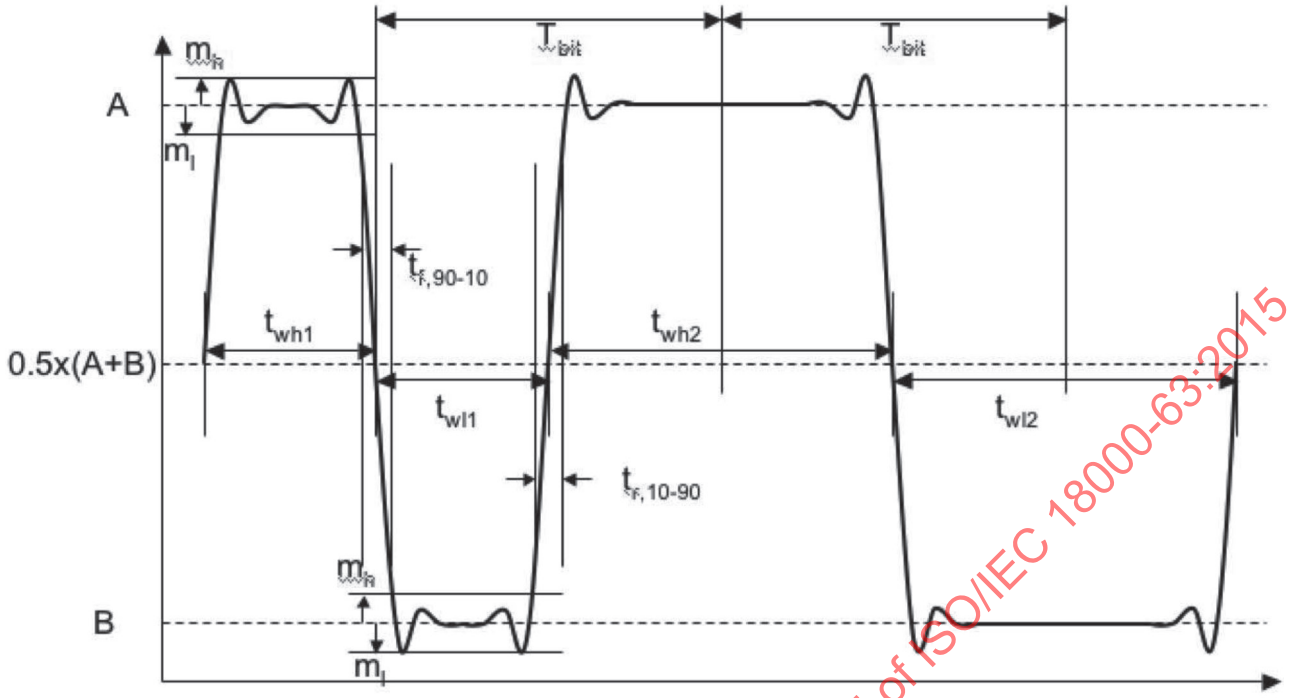


Figure 7.29 — Manchester waveform

Table 7.6 — Manchester command modulation parameters

PARAMETER	Minimum	Nominal	Maximum	Units
Tbit		1/data_rate		µs
(A-B)/A	80		90	%
mh/(A-B)	0		5	%
ml/(A-B)	0		5	%
tf,90-10/Tbit	0		33	%
tr,10-90/Tbit	0		33	%
twh1/Tbit	45		55	%
twl1 /Tbit	45		55	%
twh2 /Tbit	90		110	%
twl2 /Tbit	90		110	%

7.5.2.1.6 Interrogator to Tag normal preamble

The normal preamble consists of two parts. Both parts shall be Manchester encoded at the selected data rate indicated in the activation command. The first part is 21 bits of a pseudorandom training sequence that enables receiver dynamic range adjustment, AC coupling training, and timing acquisition. The specific sequence is a 31 bit m-sequence truncated to 21 bits and is defined by the NRZ equivalent below and the Manchester equivalent in Figure 7.30.

Normal preamble 21 bit training sequence = 1111100 0110111 0101000

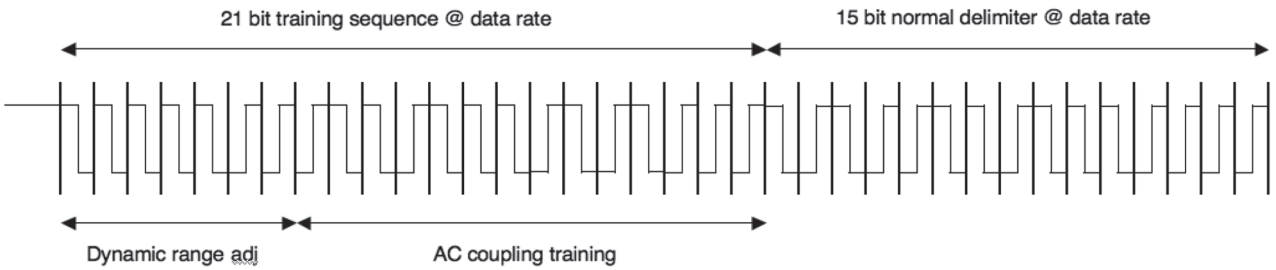


Figure 7.30 — Forward (R=>T) preamble

The second part is a 15 bit pseudo-random delimiter that uniquely indicates a normal command and the start of the command data. The specific normal delimiter sequence is a 15 bit m-sequence defined by the NRZ equivalent below and is the logical inverse of the activation delimiter.

Normal preamble 15 bit delimiter = 10100 11011 10000

The first bit of the command data shall follow immediately after the last bit of the delimiter at the same data rate as the preamble.

7.5.2.1.7 Transmission order

The transmission order for all R=>T communications shall respect the following conventions:

— Within each message, the most-significant word shall be transmitted first, and

— Within each word, the most-significant bit (MSB) shall be transmitted first.

7.5.2.1.8 Command data bit stuffing

Compliant Interrogators and Tags shall use selective bit stuffing on all command data bits that follow the preamble in both normal and *Activation* commands. Interrogators shall monitor the command data bit stream for the following 14 bit sequence defined in NRZ notation:

0101100 1000111

This specific pattern is the first 14 bits of the activation delimiter. If the pattern is matched exactly in the command data stream, a logical bit 0 shall be bit stuffed into the data stream at the end of this 14 bit sequence, extending the command by one bit for each bit stuffed. This eliminates the possibility of matching of the 15 bit activation delimiter in all Manchester encoded commands. The resulting 15 bit pattern will be as defined in NRZ notation:

0101100 1000111 0

Tags shall monitor the command data bit stream for the same 14 bit pattern with a logical zero in the 15th bit. The Tag shall destuff the logical zero bit from the data bit stream. The output of the destuffing algorithm in Tags will provide the command data format as defined in the commands summary of clause 7.5.4. If the 14 bit pattern is followed by a logical one, an activate delimiter has been received. If the Tag has already received a valid activation command and is processing command data, the command is invalid. If the invalid sequence was received at a data rate of 16 Kbps or higher, the Tag shall then ignore the invalid command. If the command was being received at 8 Kbps, the Tag shall ignore the invalid command and optionally transition to the **activation code check** state to process a possible new *Activation* command.

7.5.2.1.9 Cyclic Redundancy Check (CRC)

CRC calculation shall be the same as defined in clause 6.3.1.5. The order of processing in Interrogators assembling a command is:

1) Calculate the CRC beginning with the first bit of command data following the last bit of the preamble.

2) Process for bit stuffing up to the last bit of the CRC.

The order of processing in Tags handling a received command:

1) Execute the destuffing operation.

2) Calculate the CRC on the output of the destuffing operation.

7.5.2.1.10 Link timing

The Manchester Battery Assisted Passive link timing uses the same parameters as the passive mode in clause 6.3.1.6. However, the parameters T_1, T_2, T_3, and T_4 shall use the definitions in Table 7.7 below, with the exception for optional BLFs > 640 kHz noted below this table.

Table 7.7 — Manchester link timing parameters

Parameter	Minimum	Nominal	Maximum	Description
T_1	MIN(2 Tbit,10T_{pri}) × (1 – 2|FT|) – 2µs	MAX (2.5 Tbit, 10T_{pri})	MAX(3 Tbit,10T_{pri}) × (1 + 2|FT|) + 2µs	Time from Interrogator transmission to Tag response (specifically, the time from the end of the last bit of the Interrogator transmission to the first rising edge of the Tag response; in the case of a data 0 as the last bit, the end of the last bit is ½ bit time after the last rising edge), measured at the Tag's antenna terminals.
T_2	0.25Tbit		If T2ext=0, then 4Tbit If T2ext = 1, then max = INACT_T or Global Timeout, as selected by the Tag manufacturer	Interrogator response time required if a Tag is to demodulate the Interrogator signal, measured from the end of the last (dummy) bit of the Tag response to the first falling edge of the Interrogator transmission.
T_3	0.0T_{pri}			Time an Interrogator waits, after T_1, before it issues another command
T_4	4xTbit			Minimum time between Interrogator commands
T_5	MIN(2 Tbit,10Tpri) × (1 – 2|FT|) – 2µs		20ms	Delayed reply time from Interrogator transmission to Tag reply. Specifically, the time from the last rising edge of the last bit of the Interrogator transmission to the first rising edge of the Tag reply for a delayed Tag reply, measured at the Tag's antenna terminals.

Table 7.7 *(continued)*

Parameter	Minimum	Nominal	Maximum	Description
T_6	MIN(2 Tbit,10Tpri) × (1 – 2|FT|) – 2µs		20ms	In-process reply time from Interrogator transmission to the first Tag reply. Specifically, the time from the last rising edge of the last bit of the Interrogator transmission to the first rising edge of the first Tag reply indicating that the Tag is either (a) still working, or (b) is done, measured at the Tag's antenna terminals
T_7	MAX(250µs, T_2(max))		20ms	In-process reply time between Tag replies. Specifically, the time from the end of the last (dummy) bit of the Tag's prior transmission indicating that the Tag is still working to the first rising edge of the current Tag reply indicating that the Tag is either (a) still working, or (b) is done, measured at the Tag's antenna terminals

NOTE 1 Tbit denotes the Manchester bit time of the normal commands.

NOTE 2 A Tag may exceed the maximum value for T_1 when responding to commands that write to memory.

NOTE 3 The maximum value for T_2 shall apply only to Tags in the reply or acknowledged states.

NOTE 4 If T2ext = 0, a Tag shall be allowed a tolerance of 4Tbit < T2(max) < 8 Tbit in determining whether T_2 had expired and therefore an "invalid command" was received. This allows the back calculation of elapsed T_2 time for command validation based on a measurement of the longer time from the end of the backscatter to the end of the command preamble where command start time is definitively established. If T2ext = 1, the Tag shall allow valid responses up to the INACT_T or Global Timeout expiration, whichever is selected by the Tag manufacturer.

NOTE 5 FT is the frequency tolerance specified for Manchester Tag BLFs of 640 kHz or less (currently 4%).

Tags and Interrogators that support optional higher frequency BLFs (see Manchester *Query_BAT* command as described in sub-clause 7.5.4.3.1.1) shall have their minimum and maximum T_1 times relaxed to not have to be shorter than that calculated for the maximum required BLF of 640 kHz. This prevents unreasonably small Tag data fetch times and Tag and Interrogator TR turn-around times for Tags and Interrogators supporting these higher BLFs.

7.5.2.1.11 Transmit mask

Interrogators that are claimed to operate according to this International Standard shall meet the local regulations for out-of-channel and out-of-band spurious radio-frequency emissions.

Interrogators that are claimed to operate in the Manchester battery assisted mode, in addition to meeting the local regulations, shall also meet the Manchester mode Transmit Mask specified in this International Standard:

Manchester mode transmit mask: For Interrogator transmissions centred at a frequency f_c, a 3.125/Tbit bandwidth R_{BW} also centred at f_c, an offset frequency f_o = 3.125/Tbit, and a 3.125/Tbit bandwidth S_{BW} centred at $(n \times f_o) + f_c$ (integer n), the ratio of the integrated power $P()$ in S_{BW} to that in R_{BW} with the Interrogator transmitting random data shall not exceed the specified values:

$|n| = 1$: $10\log_{10}(P(S_{BW}) / P(R_{BW})) < -30$ dB

$|n| = 2$: $10\log_{10}(P(S_{BW}) / P(R_{BW})) < -60$ dB

$|n| > 2$: $10\log_{10}(P(S_{BW}) / P(R_{BW})) < -65$ dB

Where $P()$ denotes the total integrated power in the 3.125/Tbit reference bandwidth. This mask is shown graphically in Figure 6.7, with dBch defined as dB referenced to the integrated power in the reference channel.

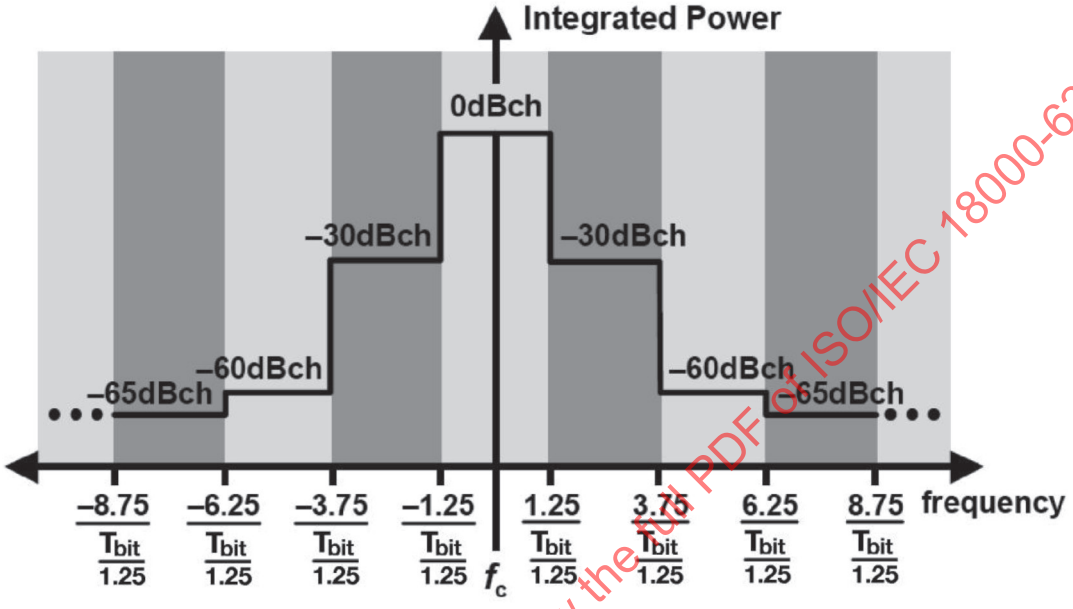


Figure 7.31 — Transmit mask for Manchester battery assisted mode

7.5.2.2 Tag-to-Interrogator (T=>R)

Tags shall communicate with Interrogators using the modulations described in 6.3.1.3 with extended values of M and with frequency accuracy of +/- 4% for BLF up to and including 640 kHz. For BLF greater than 640 kHz, the frequency accuracy shall be better than +/- 1.5%. The specific data rate is commanded by *Query_BAT* (see 7.5.4.3.1.1) using the values of M and BLF. Tags shall support all possible data rate combinations, whereas Interrogators may select the specific combinations of BLF and M to be implemented.

7.5.3 Manchester Activation

To extend battery life Type C battery assisted Manchester Tags remain in the low power **hibernate** state until receiving an *Activation* command containing a valid Activation Mask that matches all or a selected part of an "Activation Code" (AC) stored on the Tag. The use of two separate Activation Codes allows the ability to have a general purpose primary AC (typically static such as based on the Tag UII) and to have an additional application specific secondary AC (may contain for instance dynamically mapped Tag status information). The fraction of a Tag population chosen for activation may be controlled by means of partial AC matching. Non-selective activation of all battery assisted Tags within the range of the Interrogator is also supported by issuing a "Wildcard Activation".

In order to move from the low power **hibernate** state to the **battery ready** state an *Activation* command shall be sent at a fixed data rate of 8 kbit/s. This command consists of a specific Manchester battery assisted passive preamble that is distinguishable from the normal Manchester command preamble. Following the preamble, the *Activation* command may be a short format or long format command as defined in the following clauses.

Because battery assisted systems will coexist with passive systems, care should be taken to reduce the power drain of the battery assisted systems. Passive RFID Tags as defined in ISO/IEC 18000-63 Type C receive their operating power from the Interrogator (via transmitted carrier power) which limits operating distances. The application requirements of battery assisted passive (BAP) RFID devices require distances sufficiently large to make the Interrogator an unusable power source. Additionally, BAP devices must co-exist in passive environments and care must be taken to manage power drain from the battery-enabled devices. If a BAP device continually responds to unwanted passive instructions (these being commands for "other" devices) battery power will be drained extremely quickly.

BAP Tags will typically use a very low power mode in the receiver to continuously monitor for an *Activation* command. The *Activation* command uses selection criteria in the Activation Mask to awaken only those Tags for which communication is necessary to preserve battery life. Tags which reside in the RF field for BAP mode communication may be selectively activated, accessed, then placed back into their **hibernate** (or lowest power) state, and the next set of Tags selectively activated. Activated Tags may be individually returned to the **hibernate** state using the *Next* command, or may be commanded as part of a large group of Tags using the *Deactivate_BAT* command.

The *Short Activation* command enables fast singulation cycles in applications that require a minimum of protocol overhead. The *Long Activation* command enables more control over which Tags are activated, particularly in large Tag and Interrogators population environments. Both *Short Activation* and *Long Activation* commands shall use the same preamble, followed by an Activation Control field which determines the specific format of the Activation Code.

The "**hibernate**" state is defined as that state where no Session flag timers are running and the **inventoried** flags are in state A. The "**stateful hibernate**" state is defined as that state where at least one session **inventoried** flag timer is running and the **inventoried** flag is in temporary state B. Normally, the temporary state *B* is interpreted to mean that the Tag was successfully accessed within the timer period using that session/**inventoried** flag, and that the Tag should not respond to *Activation* commands aimed at that **inventoried** flag until the timer expires. However, the Tag will respond to *Activation* commands for that session aimed at temporary state B. The purpose of such activations is normally to reprogram (refresh) the timer.

Tags will usually receive *Activation* commands while in a **hibernate** state. If a Tag that is already activated receives a valid activation preamble, the Tag may optionally ignore the command or it may clear the active **inventoried** flag, **SL** flag, and countdown timer, then transition to the **activation code check** state and continue processing the *Activation* command.

See sub-clause 7.5.3.5 for detailed state machine description of the Manchester Mode.

7.5.3.1 Activation Code

Battery assisted Type C Manchester Tags may support activation based on either a single or two separate Activation Codes for more flexibility at their manufacturer's discretion.

The Activation Code is a 96-bit value specified in part or whole by the Activation Mask during the activation sequence to move the Tag from the **hibernate** state to **battery ready**. The 96-bit Activation Code initially contains all zeroes. Once the AC is programmed and the Tag is placed into the **hibernate** state, the Tag will only respond, moving into the **battery ready** state when a valid activation sequence is received and the Activation Mask contained in the *Activation* command either matches the entire stored Activation Code or a selected sub-portion of the Activation Code. In case of Tags supporting two separate ACs the issued Activation Mask needs to match either the primary or the secondary AC (or both of them) in order to activate the Tag.

The Activation Codes can be programmed with any grouping organization which provides a selected application with desired levels of selectivity in the activation process. The Activation Codes may at the option of the user be the UII (if the UII is 96 bits), the UII plus additional data (if the UII is less than 96 bits), or part of the UII (if the UII is greater than 96 bits). They may also be a custom data string that may or may not include part of the UII.

The Minimum Mask Length (MML) contains seven bits that specify the minimum amount of bits which are required in the Activation Mask to match with the Activation Code during the activation sequence. If the received value for the Activation Mask Length is less than the value stored in the MML, the incoming *Activation* command shall be ignored. If by application of the Activation Mask Length and Offset address, an overrun occurs (that being the 96th bit of the stored Activation Code) is compared and the received length counter is not exhausted, the remainder of the *Activation* command shall be ignored and the Tag shall wait in the **hibernate** state until a new *Activation* command is received. In case a Tag supports a primary and a secondary AC, a separate MML has to be implemented for each of the two ACs.

Both the MML registers and the Activation Codes shall be readable and writeable using the *OpRegister Read/Write* command in clause 7.5.4.4.1. The first 16 bit word shall contain the 7 bits of MML in the 7 lowest numbered bits of the word, MSB first, with the remaining bits of that word currently RFU. The next six 16 bit words (the Activation Code register) shall contain the AC value, which is always 96 bits. The MSB of the AC (bit index 95 starting from zero) shall be located in lowest addressed bit of the lowest addressed word of the six 16 bit words that make up the AC register. The LSB of the Activation Code (bit index zero) shall be stored in the highest address bit the highest addressed word of the six 16 bit words that make up the AC register. In the *Activation* command, the MSB of the Activation Mask that matches the AC or a section of the AC shall be transmitted MSB first.

Table 7.8 — Structure of Minimum Mask Length and Activation Code registers (stored in hidden registers described in 7.5.4.4.1)

MML register: first 16-bit word		**AC register: next six 16-bit words**					
7 bits MML value	9 bits RFU	AC (bits 95-80)	AC (bits 79-64)	AC (bits 63-48)	AC (bits (47-32)	AC (bits 31-16)	AC (bits 15-0)

The initial MML and AC values may be written during Tag manufacturing, or later by the user.

The default value for the Activation Code shall be 0.

7.5.3.2 Activation preamble

The activation preamble provides for Tag receiver dynamic range adjustment and AC-coupling training that is essential to good sensitivity in the Tag receiver. The training period is followed by a unique frame delimiter which, when matched exactly, alerts Tags to wake up parts of its circuitry required to receive and decode the complete *Activation* command and determine if the Tag has been selected for full communication. The frame delimiter is a pseudorandom sequence which cannot occur in error-free normal command data due to the use of selective 14/15 bit stuffing. Both *Short Activation* and *Long Activation* commands shall use the same preamble.

The preamble consists of three parts. All three parts shall be Manchester encoded at 8 kbit/s. The first part is 21 bits of a pseudorandom training sequence that enables receiver dynamic range adjustment, AC coupling training, and initial timing acquisition. The specific sequence is a 31 bit m-sequence truncated to 21 bits and is defined by the NRZ equivalent below and the Manchester equivalent in Figure 7.32.

Activation preamble 21 bit training sequence = 1111100 0110111 0101000

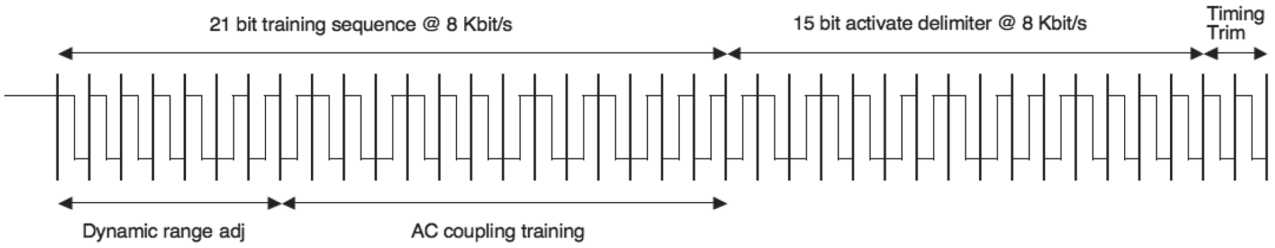


Figure 7.32 — – Activation preamble

The second part is a 15 bit pseudorandom delimiter sequence that uniquely indicates an *Activation* command and the time framing of the command. Correct reception of the delimiter portion moves the Tag from the **hibernate** state into the **activation code check** state (see 7.5.3.5) immediately following the timing trim field. The activation delimiter sequence is a 15 bit m-sequence defined by the NRZ equivalent below and is the logical inverse of the normal delimiter.

Activation preamble 15 bit delimiter = 010110010001111 (followed by two "1" bits for Timing Trim)

The third part is a timing trim field consisting of two Manchester ones and is used to finalize timing synchronization after the frame delimiter is successfully matched and the Tag clock is turned on.

The first bit of the *Activation* command shall follow immediately after the last bit of the timing trim at the fixed data rate of 8 Kbit/s.

7.5.3.3 Short Activation command

The *Short Activation* command enables fast singulation cycles in applications that require a minimum of protocol overhead. The *Short Activation* command format is shown in Figure 7.33. When matched to an internal register, the received activation command moves the Tag from its **hibernate** state to an active state (**battery ready** state in the state machine). The *Short Activation* command is composed of four fields: (in order of reception at the Tag) Activation Control, Mask Length, Offset, and Activation Mask. Each is described in detail below.

Activation Control	Mask Length	Offset	Activation Mask (0-96 bits)
(12 bits)	(7 bits)	(If MaskLength = 0 or MaskLength = 96: 0 bits, Else: 7 bits)	

Figure 7.33 — Manchester *Short Activation* command format

The Activation Control field shall be used to configure the Tag receiver post activation behaviour. The bits in the Activation Control field are defined in Table 7.9. The RFU bits shall be set to all zeros for this standard version. If the Tag decodes any non-zero bits in this RFU field it shall ignore the *Activation* command. The Activation Version indicates the short or long format of the *Activation* command being used. The Forward Data Rate indicates the rate at which subsequent commands will be sent. The Sensitivity field shall command Tags to use high or low sensitivity for Tags that are capable of two modes of sensitivity. Low sensitivity Tags are defined as capable of receiving approximately -30 dBm up to +10 dBm, with a recommended upper limit of +15 dBm. High sensitivity Tags are defined as capable of receiving approximately -30 dBm and lower signal levels. These sensitivities refer to the 8 kbps data rate, and other data rates are understood to vary. Tags are not required to implement high sensitivity, and if they do not they shall ignore the command to switch to a high sensitivity mode.

In the Wildcard Activation case of Mask Length = 0 to wake up all Tags (described below), the Offset and Activation Mask are not needed and are not transmitted. If Mask Length is set to 96, then Offset is redundant and is not transmitted.

Table 7.9 — Manchester Activation Control field description

	RFU	Activation Version	Forward Data Rate	Sensitivity	RFU
# of bits	3	1	3	1	4
Description	RFU	0: short format 1: long format Must be set to 0 for Short Activation	000: RFU 001: 8 kbit/s 010: 16 kbit/s 011:32 kbit/s 100: 64 kbit/s 101: 128 kbit/s 110: RFU 111: RFU	0: low sensitivity (approx -30 dBm to +10 dBm) 1: high sensitivity (approximately -30 dBm and below)	

The Mask Length is a required field that contains seven bits. It may contain values from zero to 2^7-1(= 127). The Mask Length specifies the length of the transmitted Activation Mask from zero bits up to and including 96-bits. AC values greater than 96 are currently unused and reserved for future use (RFU). The length field is used in conjunction with the user defined Minimum Mask Length (MML). The MML controls the minimum controls the minimum length that an *Activation* command can use for the Mask Length. If a value of Mask Length is less than the MML value or greater than 96, the Tag will ignore the rest of the *Activation* command and will stay in the **hibernate** state.

The Offset is a conditionally included field containing 7 bits that indicate the number of bits to offset from the first bit (MSB) of the Activation Code for this comparison. The Offset is not transmitted if the Mask Length is either 0 or 96; the Offset is transmitted for Mask Length between 1 and 95 inclusive.

7.5.3.3.1 *Short Activation* command processing and Wildcard

Activation processing: The Tag follows the below sequence in processing *Short Activation* commands.

A Tag begins processing the *Activation* command by first training, acquiring a slicing reference, and acquiring frame synchronization. The Tag loads the Activation Control field consisting of 3 RFU bits, Activation Version (1 bit of value zero for Short Activation), Forward Data Rate field (3 bits), post activation Sensitivity field (1 bit), and 4 more RFU bits, as they are acquired.

If the Tag then receives a Mask Length value of zero and the Tag is programmed with MML = 0, then a successful Wildcard has occurred (further information below) and the Tag enters the active **battery ready** state and prepares to process inventory commands. All **inventoried** flags are set to *A* and the **SL** flag is deasserted, and all associated timers are cleared. In the Wildcard Activation case the Offset and Activation Mask are not needed and are not transmitted.

In the case of a non-Wildcard activation attempt (received Mask Length > 0), if the Mask Length is less than the MML value (which is set somewhere from 0 to 96), for all Activation Codes implemented by the Tag, the Tag rejects the activation and remains in **hibernate**. If Mask Length is set to 96, then Offset is redundant and is not transmitted. Values of Mask Length which are greater than 96 shall cause the Tag to discontinue activation processing and remain in the **hibernate** state.

If Mask Length is equal or greater than the MML, then the Tag continues processing the Activation command and receives the Offset field (assuming Offset is transmitted due to Mask Length being less than 96). If the combination of the Mask Length and the Offset (zero if not transmitted) result in an overflow (an ending value greater than the size of the 96-bit internal mask register), the Tag shall interpret this as an error, ignore the remainder of the *Activation* command, and remain in the **hibernate** state.

If the sum of Mask Length and Offset value is less than or equal to 96, the Tag proceeds to processing of the Activation Mask, which is transmitted MSB first. The Activation Mask shall be compared bit by bit against the stored Activation Codes or Activation Code segments by using the Mask Length as the

number of bits to compare, and the Offset as the number of bits to offset from the first bit (MSB) of the Activation Codes for this comparison. The Tag shall discontinue the activation process if a mismatch in the bit by bit comparison is detected for the primary and the optional secondary Activation Code.

If the received Activation Mask fully matches one of the stored Activation Codes or indicated Activation Code segments, then the activation is successful and the Tag shall complete power-up operations and expect the beginning of an inventory round. All flags **inventoried** flags are set to *A* and the **SL** flag is deasserted, and all associated timers are cleared.

The Tag activation time T_A is the time required to finish the activation code check and fully power-up the Tag so that it is ready to receive a *Select* or *Query* command. The maximum Tag activation time is T_A = **2ms**.

Short Wildcard Activation and authorization:

If the MML register is set to zero, then the Mask Length within the *Activation* command may also be set to zero and be accepted by the Tag as "Wildcard Activation", i.e. a match for all devices in the field. This causes all receiving devices to initiate powering up into the active **battery ready** state. Short Wildcard Activation does not require any flag matching, whereas the later described Long Wildcard Activation may require flag matching. Note that if the MML for one of the supported Activation Codes is programmed to length greater than zero, this provides some security in the sense that the Tag is not authorized to accept Wildcard Activation, and the Interrogator must know at least the number of bits of Activation Code specified in the MML register in order to awaken the Tag.

In the Wildcard Activation case the Offset and Activation Mask are not needed and are not transmitted.

7.5.3.3.2 Short Activation post Activation behaviour

***Select* command behaviour for Short Activation:** The Tag is short activated in Interference Rejection = OFF (Promiscuous) mode (Session Locking not in effect) and all **inventoried** flag timers have been cleared. The Tag shall obey *Select* command reprogramming of the **SL** and **inventoried** flags. The *Select* command also moves any Tags out of the **battery ready** state back to the **battery ready** state. Upon return to **hibernate** the Tag shall set all **inventoried** flags to state *A* and the **SL** flag to the deasserted (**~SL**) state.

Manchester *Query_BAT* command behaviour for Short Activation: The Tag is short activated in Interference Rejection = OFF (Promiscuous) mode (Session Locking not in effect) and all **inventoried** flag timers have been cleared. The Tag shall respond to all *Query_BAT* commands as it normally does according to passive state machine operation (Queries take the Tag immediately into the **arbitrate** state using the session indicated in the *Query_BAT* command). Upon return to **hibernate** the Tag shall set all **inventoried** flags to state *A* and the **SL** flag to the deasserted (**~SL**) state.

7.5.3.4 Long Activation command

The *Long Activation* command enables more control over which Tags are activated, particularly in large Tag population environments. When correctly matched to Tag **inventoried** flag values, and to one of the Activation Codes or indicated Activation Code segments, the Tag shall advance from its **hibernate** state to the active state (**battery ready** state in the state machine). This will also occur for the Wildcard Activation case. The *Long Activation* command is composed of the seven fields shown in Figure 7.34.

Activation Control (12 bits)	Target (16 bits)	Mask Length (7 bits)	Offset (7 bits) (0 bits if Length=0)	Activation Mask (0 - 96 bits) (0 bits if Length=0)	Interrogator Info (17 bits)	CRC (16 bits)

Figure 7.34 — Long Manchester Activation command format

The Activation Control, Mask Length, Offset address, and Activation Mask fields shall be the same as defined in the Short Activation command.

In the Wildcard Activation case of Mask Length = 0 to wake up all Tags (described below), the Offset and Activation Mask are not needed and are not transmitted. If Mask Length is set to 96, then Offset is redundant and is not transmitted.

The CRC-16 shall be calculated by Interrogators over the first Activation Control bit to the last Interrogator Info bit. The use of the CRC-16 for Tags is optional.

The Target field is expanded as shown below.

Table 7.10 — Manchester Activation Target field description (Long Activation format only).

	Activation Tag Type Select field	**Session**	**Inventoried flag use**	**Inventoried flag target**	**Stateful hibernation timeout**
# of bits	8	2	1	1	4
Description	As specified in the Activation Tag Type Select field (Table 7.11)	00:S0 01:S1 10:S2 11:S3	0: Don't care for **inventoried** state 1: Do care for **inventoried** state	0: *A* 1: *B* So targeted if "Do Care" is specified.	0000: 0 s 0001: 0.25 s 0010: 0.5 s 0011: 1 s 0100: 2 s . . . 1111: 4,096 s

NOTE 1 The accuracy of the **hibernate** timer over the nominal temperature range shall be at least +/- 40%, and over the extended temperature range shall be at least +/- 50%.

NOTE 2 If **inventoried** flag use is set to 1: Do Care, then the Tag will upon activation exhibit "Session Locking" and generally reject commands for which the session ID does not match the "activating session" given by this table. For more information, see Annexes E and F. If Interrogator Locking is in effect, that will also set "Session Locking" into effect AFTER activation regardless of the state of **inventoried** flag use. But, even if Interrogator Locking is in effect, the **inventoried** flag use bit will control the Activation, meaning that if Don't Care is selected on **inventoried** flag use, then Tags will activate regardless of Session flag state.

NOTE 3 If **inventoried** flag use = Don't Care, then upon transition to the **battery ready** state the Tag clears all timers and makes the associated **inventoried** flags available to any command that uses them that the Tag receives.

Table 7.11 — Manchester Activation Tag Type Select Field (NOTES 1-5)

Interpretation (NOTE 1)	**Sensor Alarm**	**Full Function Sensor**	**Simple Sensor**	**RFU**	**RFU**	**RFU**	**Battery Assisted Passive** NOTE 4
1	1	1	1	1	1	1	1
0: Inclusive 1: Exclusive	0: No 1: Yes	0: No 1: Yes	0: No 1: Yes	0	0	0	0: No 1: Yes

NOTE 1 This bit dynamically switches the interpretation of the remainder of the Tag Type Select field between "Inclusive" (the Tag responds if it matches any selection marked "Yes") and "Exclusive" (the Tag responds only if it matches all selected criteria marked "Yes").

NOTE 2 In this table, "Yes" means that category of Tag is considered for inclusion in the activation. In the "Inclusive" case for a criteria listed as Yes the Tag will respond if it meets other activation criteria (such as AC). In the "Exclusive" case the Tag responds only if the Tag meets all criteria marked Yes. Logically the Tag responds to "Inclusive" if a logical OR function of all marked Yes criteria = 1. The Tag responds to "Exclusive" if a logical AND of all marked Yes criteria = 1.

NOTE 3 RFU bits shall be set to zero until defined. If the Interpretation field is "0, Inclusive" the Tag shall respond if it matches at least one selected criteria (the Tag ignores the RFUs because the implied logical OR function is still a one). If the interpretation field is "1, Exclusive", the Tag shall NOT respond if any RFU bits are non-zero (in that case the Tag does not know if it matches the implied AND function).

NOTE 4 This field means the Tag supports a Manchester receiver and a backscatter transmitter.

NOTE 5 The Manchester Activation Tag Type Select Field is identical to the Manchester *Query.BAT* Tag Type Select field.

The **stateful hibernate** timeout field determines the timeout period of the session **inventoried** flag timer specified in the session field.

The Interrogator Info field provides information as to Interrogator identity, whether the Tag is to reply to only the activating Interrogator or not after wake up, and the regulatory region of operation. The regulatory region field is an optional field for Interrogators and Tags and the definition is TBD.

Table 7.12 — Manchester Activate Interrogator Info Field Description

	Interrogator ID	**Interrogator Lock**	**Regulatory region**
# of bits	8	1	8
Description	Interrogator ID code	0: Tag allows any Interrogator to access 1: Tag only allows this Interrogator to access	Specifies a region in which the Tag operates. Definition TBD.

7.5.3.4.1 Long Activation processing and Wildcard

Long Activation processing: The Tag follows the sequence below in processing *Long Activation* commands.

A Tag begins processing the *Long Activation* command by first training, acquiring a slicing reference, and acquiring frame synchronization. The Tag loads the Activation Control field as it is acquired. The Tag then loads the activation Target field up to the flag use field. If the **inventoried** flag Care / Don't Care states as transmitted in the *Long Activation* command do not match those in the Tag, the Tag discontinues activation processing and remains in **hibernate** or **stateful hibernate**, awaiting the next *Activation* command.

With matching **inventoried** flag states, the Tag loads the rest of the activation Target field, including the Tag Type Select field and its set of criteria that are evaluated inclusively or exclusively as indicated. The Tag may discontinue processing the *Activation* command if it does not match the indicated criteria. If the Tag receives a Mask Length value of zero and the Tag is programmed with MML = 0 for all of its Activation Codes, then a wildcard attempt is in progress (further information below) and the Tag continues activation processing. In the Wildcard Activation case the Offset and Activation Mask are not needed and are not transmitted, so the next field received is the CRC-16. If the CRC-16 is passed (or the Tag does not use it), the Tag performs a final check on the Session flag state and if it still matches (has not timed out), it enters normal active mode and prepares to respond to normal inventory commands. Flag states remain unchanged and any timers in operation continue to run. If the CRC-16 is not passed the Tag remains in the **hibernate** state.

In the case of a non-Wildcard activation attempt with the **inventoried** flag state matching (received Mask Length > 0), if the Mask Length is less than the MML value (which is set somewhere from 0 to 96) for all Activation Codes implemented by the Tag, the Tag rejects the activation and remains in **hibernate**. If Mask Length is set to 96, then Offset is redundant and is not transmitted. Values of Mask Length which are greater than 96 shall cause the Tag to discontinue activation processing and remain in the **hibernate** state.

If Mask Length is equal or greater than the MML, the Tag continues processing the *Activation* command and receives the Offset field (assuming Offset is transmitted due to Mask Length being less than 96). If the combination of the Mask Length and the Offset (zero if not transmitted) results in an overflow (an ending value greater than the size of the 96-bit internal mask register), the Tag shall interpret this as an error, ignore the remainder of the *Activation* command, and remain in the **hibernate** or **stateful hibernate** state.

If the sum of Mask Length and Offset value is less than or equal to 96, the Tag starts processing of the Activation Mask, which is transmitted MSB first. The Activation Mask shall be compared bit by bit against the stored Activation Codes or Activation Code segments by using the Mask Length as the number of bits to compare, and the Offset as the number of bits to offset from the first bit (MSB) of the Activation Code for this comparison. The Tag shall discontinue the activation process and remain in **hibernate** if any mismatch in the bit by bit comparison is detected.

If the received Activation Mask fully matches one of the stored Activation Codes or indicated Activation Code segments, the Tag will continue to processing of the CRC-16. If the CRC-16 is passed (or the Tag does not use it), then the Tag performs a final check on the Session flag states and if it still match (has not timed out), then the activation is successful and the Tag shall complete power-up operations and expect the beginning of an inventory round. If the CRC-16 is failed, then the Tag discontinues activation processing and remains in **hibernate**, awaiting the next *Activation* command.

The Tag activation time T_A is the time required to finish the activation code check and fully power-up the Tag so that it is ready to receive a *Select* or *Query* command. The maximum Tag activation time is T_A = **2ms**.

Long Wildcard activation and authorization:

If the MML register is set to zero, then the Mask Length within the *Activation* command may also be set to zero and be accepted by the Tag as "Wildcard Activation", i.e. a match for all devices in the field for which the specified **inventoried** flag also match. If the Wildcard Activation is to apply to all Tags, then the Inventoried flag usage field is set to the "Don't Care" state. Note that if the MML for one of the supported Activation Code is programmed to length greater than zero, this provides some security in the sense that the Tag is not authorized to accept Wildcard Activation, and the Interrogator must know at least the number of bits of Activation Code specified in the MML register in order to awaken the Tag.

In the Wildcard Activation case the Offset and Activation Mask are not needed and are not transmitted. The CRC-16 (if used by the Tag) is still applicable to the Wildcard and is transmitted.

7.5.3.4.2 Long Activation post Activation behaviour

7.5.3.4.2.1 Long Activation flag timeout definition, behaviour, and usage

Manchester flag timeout definition:

Manchester **SL** and **inventoried** flag persistence shall have a different flag behaviour from passive and BAP PIE specified as follows. The definition of flag persistence shall change from beginning upon Interrogator transmissions dropping below the sensitivity of the Tag or the loss of decoding to beginning at the time the Tag transitions from normal active mode back to **hibernate**. These new flag persistences shall be referred to as "Manchester flag timeout and shall have accuracy as follows:

+/- 40% over the nominal temperature range of -25 to +40 °C.

+/- 50% over the extended temperature range of -40 to +65 °C.

Upon deactivation and return to the **hibernate** state, all flags without active timeouts in operation shall return to state A. Flags with active timeouts associated with other sessions that are in state *B* shall continue their timeouts (i.e. **stateful hibernate**), and upon timeout shall reset to state A. Flags associated with the current session (those specified in the last activation) that are in state *B* shall begin their timer operation and revert to state *A* upon timeout. Flags with active timeouts for which the state is already *A* upon entry to **hibernate** shall remain in *A* and need not continue or begin timer operation. Session flags with active timeouts and not used in a current activation session shall continue to countdown while the activation session is on-going.

When Manchester Tags are operating in BAP PIE mode, they may support the passive PIE compatible flag persistence times as given in Table 7.1. These are identical to the passive flag persistences of Table 6.20, except that they have specific maximums for the **S2**, **S3**, and **SL** flags and delay the beginning of flag persistence timeout by INACT_T. Alternatively, Manchester Tags that are operating in BAP PIE mode may support the passive flag persistences of Table 6.20. When a Manchester Tag is operating in Dead Battery Response mode, it may continue to support the passive compatible persistences of Table 7.1, or it may revert to the more relaxed persistences of Table 6.20.

Manchester Tags may but are not required in BAP PIE mode to use environmental validation with regards to INACT_T or Selective Global Timeout refresh. Alternatively they may use signal strength above a threshold to refresh INACT_T or Selective Global Timeout. If a Manchester Tag prepares to participate in a BAP PIE interrogation round, it shall set its **inventoried** flags to *A* and deassert the select flag. Any Manchester Hibernation timers in operation continue to operate in the background. When the Tag returns to Manchester **hibernate** it shall set its **inventoried** flags according to the state of its **hibernate** timers and deassert the select flag.

When a Manchester Tag is operating in Dead Battery Response mode, it may continue to support the passive compatible persistence maximums of Table 7.1, or it may revert to the more relaxed persistences of **Manchester flag behaviour and usage**.

Manchester flag behaviour and usage:

inventoried flag states in state **hibernate** (full or **stateful**) have the typical meaning *A* for "Default / Not Recently Inventoried" (timer not running) and *B* for "Temporary / Recently Inventoried" (timer running). System control in Normal Mode would normally also use *A* for "Not Inventoried" and *B* for "Inventoried".

Each session/**inventoried** flag shall have its own individual timer for Manchester mode. To program different **inventoried** flags to different timeout values, it is necessary to use multiple *Activation* commands with Session Locking. In Normal Mode with Session Locking the **inventoried** flag of the activating session indicates inventory status in that session, and the other **inventoried** flags indicate their timer status.

With Session Locking not in effect, the Manchester Tag shall upon Activation clear any timers, set all **inventoried** flags to A, and set the **selected** flag to deasserted. Upon return to **hibernate** it shall set all **inventoried** flags to *A* and **selected** flag to deasserted.

With Session Locking in effect, a Tag that returns to **hibernate** via INACT_T or (Selective) Global Timeout shall clear the timer associated with activating session, and set that **inventoried** flag to A. This does not affect any other Hibernation mode timers that may be in operation.

With Session Locking in effect, Tags shall carry their inventory state from **hibernate** to Normal Mode. If returned to **hibernate** via the *Deactivate_BAT* or *Next* commands, they shall set **inventoried** flags to *B* if they have associated timers running and to *A* if no timer is currently running on that **inventoried** flag.

With Session Locking in effect, the Tag shall not respond to mismatched (indicated session does not match activation session) commands. Activation Session, as passed in the Activation command, is a variable that Tags implementing **hibernate** must keep track of in order to determine if there is a session match.

Thus, with Session Locking in effect the Tag is not able to participate in Query rounds with session mismatched Interrogators without first going to **hibernate** and being reactivated by the Interrogator

using a different session. Only if Session Locking is not in effect can the Tag go into an inventory round with Interrogators using different sessions during one period of Normal Mode operation.

If a Manchester Tag receives a successful *Long Activation* command that reprograms an active timer, it shall clear the referenced timer, reset it with the newly received time value, and then proceed to the Normal Mode. Typically this operation is performed to "refresh" a timer that is about to expire, and the Tag or Tags would then be deactivated to return to **hibernate** mode, without being reinventoried, with the new timer value and **inventoried** flag in state B. The *Long Activation* command can only program or reprogram a single **inventoried** flag timer per command.

Examples of behaviour of these flags as the Tag crosses from **hibernate** to Normal Mode and back are given in Table 7.13 in Manchester sub-clause 7.5.3.4.2.1. This table applies to Manchester (Short and Long activation cases) in the row relative to Long Activation. See also Figure 7.35 for the Tag extended state machine, and sub-clause 7.5.4.3.7.1 for details of the Manchester *Deactivate_BAT* command and its effect on state machine state and flag values.

Examples of behaviour of these flags as the Tag crosses from **hibernate** to Normal Mode and back are given in Table 7.13 below. This table applies to Manchester (Short and Long activation cases). See also Figure 7.35 for the Tag extended state machine, and sub-clause 7.5.4.3.7.1 for details of the Manchester *Deactivate_BAT* command and its effect on state machine state and flag values.

Table 7.13 — Manchester inventory flag behaviour in Hibernation and Normal Modes

	State/command sequence	Next state	S0	S1	S2	S3
1	Hibernation		*A*	*A*	*A*	*A*
1.1	After Short Activation OR Long Activation with Don't care for **inventoried** state (no Session Locking)	**battery ready**	*A*	*A*	*A*	*A*
1.1.1	After *Query_BAT* on S1, successful singulation, *Next* -> **hibernate**	**hibernate**	*A*	*A*	*A*	*A*
1.1.2	After *Query_BAT* on S1, successful singulation, *QueryRep*	**battery ready**	*A*	*B*	*A*	*A*
1.1.2.1	After *Deactivate_BAT* (any session, but **SL** and **inventoried** flag state pointed do match)	**hibernate**	*A*	*A*	*A*	*A*
1.1.2.2	After *Deactivate_BAT* (session = S1, **SL** match, but **inventoried** flag of S1 Target = *A* does not match)	**battery ready**	*A*	*A*	*A*	*A*
1.1.2.2	After INACT_T or (Selective) Global Timeout	**hibernate** (including brief stay in **stateful hibernate** to check timers expired)	*A*	*A*	*A*	*A*
1.1.3	After INACT_T or (Selective) Global Timeout (no successful singulation)	**hibernate**	*A*	*A*	*A*	*A*
1.2	After Long Activation with Do care for **inventoried** state S1=A (Session Locking)	**battery ready**	*A*	*A*	*A*	*A*
1.2.1	After *Query_BAT* on S1, successful singulation, *Next* -> **hibernate**	**stateful hibernate**	*A*	*B*	*A*	*A*
1.2.2	After *Query_BAT* on S1, successful singulation, *QueryRep*	**battery ready**	*A*	*B*	*A*	*A*
1.2.2.1	After *Deactivate_BAT*	**stateful hibernate**	*A*	*B*	*A*	*A*
1.2.2.2	After INACT_T or (Selective) Global Timeout	**stateful hibernate**	*A*	*A*	*A*	*A*

Table 7.13 *(continued)*

	State/command sequence	Next state	S0	S1	S2	S3
1.2.3	After INACT_T or (Selective) Global Time-out (no successful singulation)	**hibernate**	*A*	*A*	*A*	*A*
2	**Stateful hibernation**, S0 & S3 timers running		*B*	*A*	*A*	*B*
2.1	After Short Activation (no Session Locking)	**battery ready**	*A*	*A*	*A*	*A*
2.1.1	After *Query_BAT* on S1, successful singulation, *Next* -> **hibernate**	**hibernate**	*A*	*A*	*A*	*A*
2.1.2	After *Query_BAT* on S1, successful singulation, *QueryRep*	**battery ready**	*A*	*B*	*A*	*A*
2.1.2.1	After *Deactivate_BAT*	**hibernate**	*A*	*A*	*A*	*A*
2.1.2.2	After INACT_T or (Selective) Global Time-out	**hibernate**	*A*	*A*	*A*	*A*
2.1.3	After INACT_T or (Selective) Global Time-out (no successful singulation)	**hibernate**	*A*	*A*	*A*	*A*
2.2	After Long Activation with Don't care for **inventoried** state (no Session Locking)	**battery ready**	*A*	*A*	*A*	*A*
2.2.1	After *Query_BAT* on S1, successful singulation, *Next* -> **hibernate**	**hibernate**	*A*	*A*	*A*	*A*
2.2.2	After *Query_BAT* on S1, successful singulation, *QueryRep*	**battery ready**	*A*	*B*	*A*	*A*
2.2.2.1	After *Deactivate_BAT* (matching session, **SL**, and **inventoried** flag state)	**hibernate**	*A*	*A*	*A*	*A*
2.2.2.2	After *Deactivate_BAT* (matching session, **SL**, but non-matching **inventoried** flag state)	**battery ready**	*A*	*A*	*A*	*A*
2.2.2.3	After INACT_T or (Selective) Global Time-out	**hibernate**	*A*	*A*	*A*	*A*
2.2.3	After INACT_T or (Selective) Global Time-out (no successful singulation)	**hibernate**	*A*	*A*	*A*	*A*
2.3	After Long Activation with Do care for **inventoried** state S1=A (Session Locking in effect)	**battery ready**	*B*	*A*	*A*	*B*
2.3.1	After *Query_BAT* on S1, successful singulation, *Next* -> **hibernate**	**stateful hibernate**	*B*	*B*	*A*	*B*
2.3.2	After *Query_BAT* on S1, successful singulation, *QueryRep*	**battery ready**	*B*	*B*	*A*	*B*
2.3.2.1	After *Deactivate_BAT* (matching session, **SL**, and **inventoried** flag state)	**stateful hibernate**	*B*	*B*	*A*	*B*
2.3.2.2	After INACT_T or (Selective) Global Time-out	**hibernate**	*B*	*A*	*A*	*B*
2.3.3	After INACT_T or (Selective) Global Time-out (no successful singulation)	**stateful hibernate**	*B*	*A*	*A*	*B*
2.4	After Long Activation with Do care for **inventoried** state S3=B (timer refresh) (Session Locking in effect)	**battery ready**	*B*	*A*	*A*	*B*
2.4.1	After *Deactivate_BAT* (matching session, **SL**, and **inventoried** flag state)	**stateful hibernate**	*B*	*A*	*A*	*B*

Table 7.13 *(continued)*

	State/command sequence	Next state	S0	S1	S2	S3
2.4.1.1	After S0 timer expires	**stateful hibernate**	*A*	*A*	*A*	*B*
2.4.1.2	After S0 and S3 timer expires	**hibernate**	*A*	*A*	*A*	*A*
2.4.2	After *Deactivate_BAT* (matching session, but non-matching **SL**, and matching **inventoried** flag state) NOTE: The **SL** flag is not affected	**battery ready**	*B*	*A*	*A*	*B*
2.4.3	After *Deactivate_BAT* (matching session, non-matching **inventoried** flag state, both **SL** states) NOTE 1 The S3 timer is also cleared NOTE 2 The **SL** flag is not affected	**battery ready**	*B*	*A*	*A*	*A*

7.5.3.4.2.2 Long Activation post Activation command behaviour

***Select* command behaviour without Session Locking and without Interrogator to Tag Locking:** In the case where a Tag is activated with "Don't Care" on the **inventoried** flag use bit of Table 7.10 (Promiscuous Mode), all **inventoried** flag timers have been cleared and the Tag shall obey *Select* command reprogramming of the **SL** and **inventoried** flags. The *Select* command also moves any Tags out of the **battery ready** state back to the **battery ready** state. Upon return to **hibernate** the Tag shall set all **inventoried** flags to state *A* and the **SL** flag to the deasserted (**~SL**) state.

Manchester *Query_BAT* command behaviour without Session Locking and without Reader to Tag Locking: In the case where a Tag is activated with "Don't Care" on the **inventoried** flag use bit of Table 7.10 (Promiscuous Mode), all **inventoried** flag timers have been cleared and the Tag shall respond to all *Query* commands as it normally does according to passive state machine operation (Queries take the Tag immediately into the **arbitrate** state using the session indicated in the *Query* command). Upon return to **hibernate** the Tag shall set all **inventoried** flags to state *A* and the **SL** flag to the deasserted (**~SL**) state.

From a functional point of view, it is not necessary that Session Locking be maintained when Interrogator to Tag locking is used. But, for design simplicity it is assumed that if Interrogator to Tag locking is in effect, then Session locking shall also be in effect.

***Select* command behaviour with Session Locking:** In the case where a Tag is activated with "Do Care" on the **inventoried** flag use bit of Table 7.10 (Interference Rejection Mode ON), the Tag shall obey *Select* command **SL** or **inventoried** flag programming only if the Target of the *Select* command is **SL** or the session that matches that of the Activation Session. If Interrogator Locking is in effect, the Tag shall also verify that the Interrogator ID field is included in the *Select* command and that the ID value matches that of the last *Activation* command before processing the *Select* command. *Select* commands with non-matching Session ID's or Interrogator ID's shall be ignored. Upon return to **hibernate** the Tag shall set the activation **inventoried** flag to *B* and commence the activation programmed timeout, and set the **SL** flag to the deasserted (**~SL**) state. Other session timers will continue their operation as previously programmed (flag state *B* while running, and flag state *A* after timeout).

Manchester *Query_BAT* command behaviour with Session Locking: In the case where a Tag is activated with "Do Care" on the **inventoried** flag use bit of Table 7.10 (Interference Rejection Mode ON), the Tag has only programmed the timer associated with the Activation Session. The Tag shall respond to *Query_BAT* commands as it normally does according to passive state machine operation (Queries take the Tag immediately into the **arbitrate** state using the **inventoried** flag indicated in the *Query* command) only if the *Query_BAT* command session matches that of the Activation Session, and for Interrogator Locking in effect, only if the Interrogator ID field of the *Query_BAT* command matches that of the last *Activation* command. Non-matching *Query_BAT* commands are ignored. Upon return to **hibernate** the Tag shall set the Activation Session **inventoried** flag to *B* and commence the activation programmed

timeout, and set the **SL** flag to the deasserted (~**SL**) state. Other session timers will continue their operation as previously programmed (flag state *B* while running, and flag state *A* after timeout).

7.5.3.5 Manchester extended Tag state machine

Tag manufacturers may define their own state control to minimize power consumption when using *Activation*. A typical state flow for Tags implementing activation mechanism is shown in Figure 7.35. This suggested state diagram is useful for standardization description and to define Tag behaviour when viewed externally. However, the actual Tag behaviour requirements are for the Tag to respond as if it is executing the states and functions as shown. The Tag activation time T_A is the time required to fully power-up the Tag after the activation code check so that it is ready to receive a *Select* or *Query_BAT* command. The maximum Tag activation time is $\mathbf{T_A = 2\ ms}$.

The states of this figure are defined as follows:

hibernate state: A low power low data rate state that allows for battery life extension as compared to the Normal Mode. In the **hibernate** state the inventory flag timers are expired. The Tag is listening for valid *Activation* commands while in this state. In this state **inventoried** flags are set to *A* and **SL** is deasserted. Note that Validated INACT_T is required for Manchester, and that it escapes the "Interference Trap" created by an interfering source and will bring a Tag back to **hibernate** in the absence of valid Manchester commands or preambles. The optional Global Timeout may also be used to escape an interference trap created by valid Manchester commands that are not applicable to a particular Tag, such as a Tag whose matching criteria for a "valid" Manchester command is not very selective.

stateful hibernate: This state is similar to **hibernate** with the extension that inventory timers are either in operation or the Tag is very briefly checking that they are in operation. When this check is complete the Tag goes to the **hibernate** state. While in **stateful hibernate** the Tag is still listening for valid *Activation* commands, and (unless there only briefly for timer state check) at least one inventory timer is operating.

After activation and inventory, Tags are typically deactivated by the *Deactivate_BAT* or *Next* commands in order to return to the **hibernate** or **stateful hibernate** state. If the Tag does not for whatever reason receive one of these two commands, it shall via either the INACT_T or Global Timeout timers return to the **hibernate** mode (see 7.5.3.6).

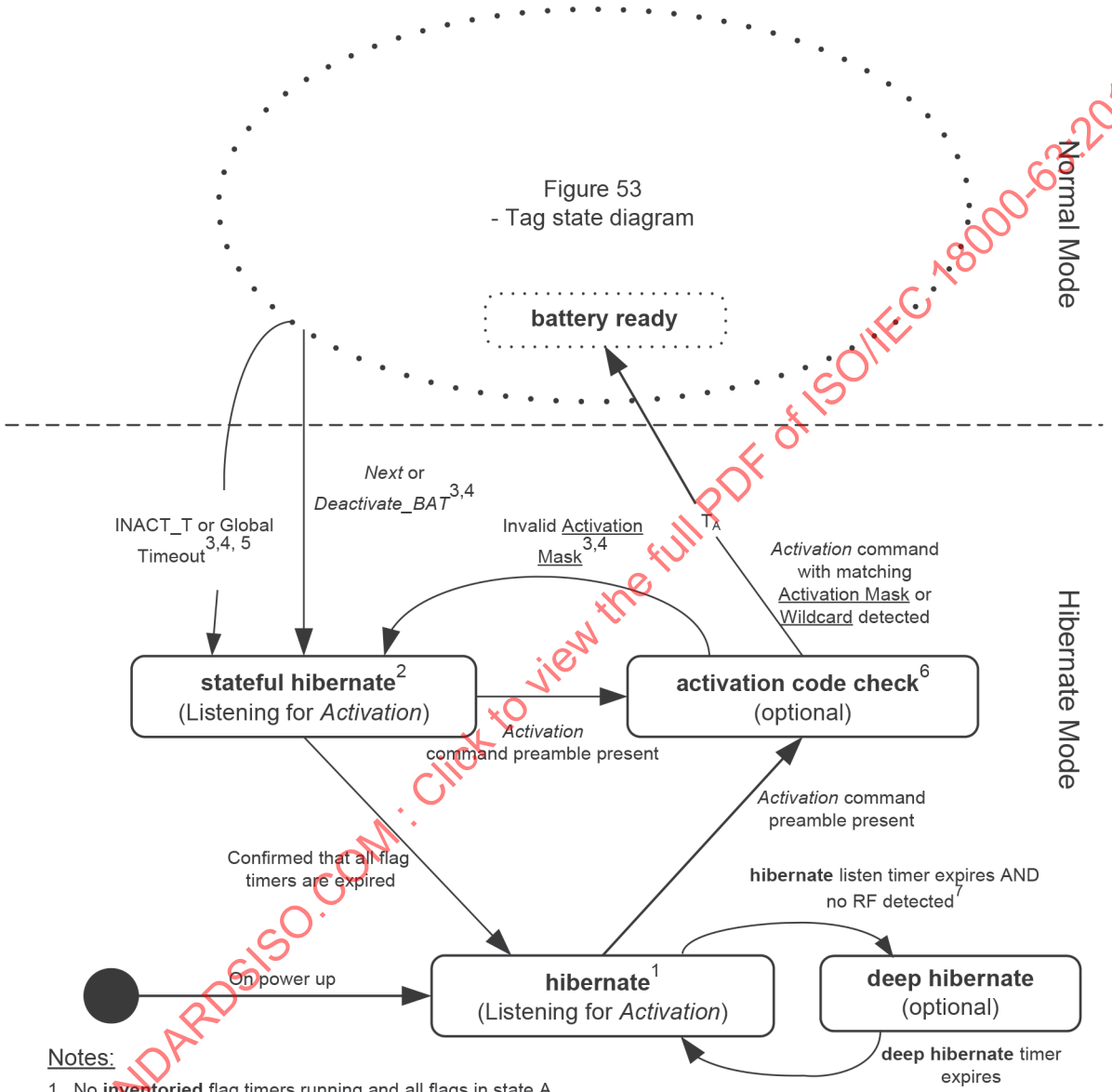
Activation code check state: In this optional state the Tag has received a valid Manchester *Activation* preamble and is proceeding to process the Activation Code. This operation can be conducted also in **hibernate** or **stateful hibernate**, but a separate state is defined to acknowledge that the Tag may be in the process of conduction power-up operations that it is not normally performing in **hibernate**. At the conclusion of successful Activation Code processing the Tag progresses to the **battery ready** state of the Normal Mode. If the AC check fails, the Tag may progress to **stateful hibernate** to continue timer operations or for brief timer state check. Or, if the Tag conducts timer check in the **activation code check** state it may progress straight to **hibernate**.

Deep hibernation state: This is an optional state used for optional duty cycling of the **hibernate** listen ation to further extend battery life. The Tag is not listening for *Activation* commands in the **deep hibernate** state, but at manufacturer option it may be listening for RF signal presence to bring it back to **hibernate** earlier than its timer dictates. The duty cycle between **hibernate** and **deep hibernate** may be fixed or user programmable. If the Tag supports this functionality, then it keeps internal timers for **hibernate** listen time and **deep hibernate** time. The Tag does not progress from **stateful hibernate** to **deep hibernate** since it has been recently accessed if in **stateful hibernate**. The Tag progresses from **hibernate** to **deep hibernate** when its **listen** Timer has expired and it has not detected either RF or valid *Activation* commands while in **hibernate**. Tags that implement the **deep hibernate** state are required to extend their **hibernate** listen timer to at least 4s when the RF level is above a manufacturer defined threshold.

It is at the Tag manufacturer's option if duty cycling of hibernation is supported, and if so, if it is fixed or programmable. If the **hibernate-deep hibernate** duty cycle is fixed the period shall not exceed 4s. If programmable, the initial **deep hibernate** period programmed by the manufacturer shall not exceed 4s, though it may be programmed deliberately longer by the user.

It is also at the Tag manufacturers' option if either current or recently detected RF or validated *Activation* commands alter the **hibernate** listen time and the **deep hibernate** time. It may be desirable to allow Tag design to go to shorter duty cycles in such situations where *Activation* commands are more likely, and to go to longer duty cycles when such commands are unlikely.

It is also at the Tag manufacturers' option if duty cycle, either fixed or programmable, may be altered by the Tag in response to a low battery state in order to extend battery life.



Notes:

1. No **inventoried** flag timers running and all flags in state A.
2. At least one **inventoried** flag timer running and **inventoried** flag in temporary state B.
3. **Stateful hibernate** means that at least one timer is running or that the tag is briefly checking if any are running.
4. The tag manufacturer has the option of checking if timers are running in the prior state and going straight to **hibernate** if none are, or first transitioning to **stateful hibernate** to check if timers are running, and then going to **hibernate**.
5. The INACT_T and (Selective) Global Timeout clear all timers and the inventory flags in **hibernate** shall be set to A.
6. **Activation code check** is an optional state that may be used if power-up sequencing begins upon successful preamble detection and not upon completed Activation Code validation.
7. The tag may extend its **hibernate** listen time if RF is detected, according to manufacturer choice. The tag may also extend hibernate listen time and/or shorten **sleep** time according in adaptation to recent activity such as *Activation* commands detected within a Last Activation Timer period (Last_Act_T).

Figure 7.35 — Extended state machine of BAP Tag implementing Manchester mode

7.5.3.6 Manchester mode Deactivation

After an Interrogator identified a battery assisted Tag and no longer needs to access it, the Interrogator shall use the *Next* or *Deactivate_BAT* command to put the Tag back into the **hibernate** state or the **stateful hibernate** state with a shorter delay. **Hibernate** commands *Next* and *Deactivate_BAT* do not flip **inventoried** state, they set state to *A* or *B* as a function of Activation Session and timer state, that is, upon return to **hibernate** the flag will be in state *A* if Session Locking is not in effect or there is no timer operation, and if Session Locking applies then in state *B* until the expiration of any programmed timer function for that flag.

There are two methods to ensure that a battery assisted Tag falls into the **hibernate** state automatically if the Tag misses the *Deactivate_BAT* or *Next* commands issued by an inventorying Interrogator. These are:

1. A Tag may automatically fall back into the **hibernate** state after a period of inactivity greater than the manufacturer defined Inactivity Threshold INACT_T, which is allowed to be 50 ms or greater. Inactivity means that a Tag does not receive valid selection commands, commands that cause it to participate in an inventory round, commands that control an inventory round the Tag participates in, or commands that directly address the Tag (Tag access commands). The Tag manufacturer may use various tests as to whether a command is valid, as function of Tag state. For example, in the **reply**, **acknowledged**, **open**, and **secured** states the Interrogator uses a Tag generated RN16. The Tag may then require the use of that RN16 as a validator to consider the command valid for the purpose of refreshing INACT_T. See sub-clause 7.3.2.2 for more information on the INACT_T timer function.

2. Tags may implement a Global Timeout in one of two forms, the standard Global Timeout and the Selective Global Timeout. The term (Selective) Global Timeout refers to either of Global Timeout or Selective Global Timeout. The standard Global Timeout causes Tags to fall back into the **hibernate** state a certain time after they were activated. This timeout, if implemented, will force the Tag back to **hibernate** despite the fact that it may be involved in a legitimate inventory round. In order to reduce the odds of an intended inventory round from being interrupted, this Global Timeout if fixed shall be 4 seconds or more (4 s minimum at 40% accuracy over nominal temperature range of -25 to + 40 °C, and 50% accuracy over extended temperature range of -40 to +65 °C). The Selective Global Timeout may refresh its timer upon reception of selected valid Manchester commands, and it may optionally use Session Locking match or Interrogator Locking match. The Selective Global Timeout, if fixed, shall be 2 seconds or more. See sub-clause 7.3.2.3 for more information on the (Selective) Global Timeout function.

Manchester Tags shall implement at least one of INACT_T, or (Selective) Global Timeout. The Tag manufacturer also has the option of allowing these timeouts to be user programmable. Programmable Tags may allow shorter limits on (Selective) Global Timeout than listed above.

Expiration of either of these timers in a Manchester Tag shall force the Tag back to **hibernate** with the timer from the activating session cleared, the **inventoried** flag state of the activating session set to A, and **selected** flag (**SL**) deasserted.

When a Manchester Tag is operating in BAP PIE mode it shall apply the delay specified in INACT_T to the start of persistence flag timeout as specified in Table 7.1. It may support the accurately specified maximums of this table, or it may use the more relaxed persistence maximums of Table 6.20.

7.5.4 Commands summary

7.5.4.1 Commands summary table

Table 7.14 lists the normal Manchester "battery-assist" commands as compared to ISO/IEC 18000-63 Type C. For Manchester commands that are different from the passive mode PIE commands in Clause 6, the differences are detailed in the clauses below that define the new or modified commands. For commands that are functionally unchanged other than the preamble and bit stuffing of the command data as defined in section 7.5.2 and 7.5.3, Interrogators and Tags shall comply with the requirements in the appropriate sub-clauses in Clause 6.

The *Activation* command is not included in this section as it is defined in clause 7.5.3 with different physical and functional characteristics from normal commands.

Table 7.14 — Manchester version commands

Command	Binary command code	Command code length	Manchester different from PIE?	Mandatory for Manchester?
QueryRep	00	2	Yes	Yes
ACK	01	2	No	Yes
Query	1000	4	n/a	Not used
QueryAdjust	1001	4	Yes	Yes
Select	1010	4	Yes	Yes
RFU	1011	4	n/a	n/a
NAK	1100 0000	8	Yes	Yes
ReqRN	1100 0001	8	No	Yes
Read	1100 0010	8	No	Yes
Write	1100 0011	8	No	Yes
Kill	1100 0100	8	No	Yes
Lock	1100 0101	8	No	Yes
Access	1100 0110	8	No	No
BlockWrite	1100 0111	8	No	No
BlockErase	1100 1000	8	No	No
BlockPermalock	1100 1001	8	No	No
Deactivate_BAT	1100 1010	8	n/a	Yes
Next	1100 1011	8	n/a	Yes
Query_BAT	1100 1100	8	n/a	Yes
Broadcast ID	1100 1101	8	n/a	No
Multirate_Reset	1100 1110	8	n/a	Yes
BAP PIE Flex Query	1100 1111	8	n/a	n/a
OpRegister Read/Write	1101 0000	8	n/a	Yes
BroadCastSync	1101 0001	8	n/a	No
ReadBuffer	**1101** 0010	8	No	No
FileOpen	**1101** 0011	8	No	No
Challenge	**1101** 0100	8	Yes	No
Authenticate	**1101** 0101	8	No	No
SecureComm	**1101** 0110	8	No	No
AuthComm	**1101** 0111	8	No	No
Reserved for future use	**1101** 1000	8	-	-
Used by ISO 18000-63	**1101** 1001	8	-	-
Reserved for future use	1101 1010 ... 1101 1111	8	-	-

Table 7.14 *(continued)*

Command	Binary command code	Command code length	Manchester different from PIE?	Mandatory for Manchester?
Reserved for custom commands (vendor-specific)	1110 0000 0000 0000 ... 1110 0000 1111 1111	16	-	-
Reserved for proprietary commands	1110 0001 0000 0000 ... 1110 0001 1111 1111	16	-	-
Untraceable	**1110**0010 00000000	16	No	No
FileList	**1110**0010 00000001	16	No	No
KeyUpdate	**1110**0010 00000010	16	No	No
TagPrivilege	**1110**0010 00000011	16	No	No
FilePrivilege	**1110**0010 00000100	16	No	No
FileSetup	**1110**0010 00000101	16	No	No
Extended commands (reserved for future use)	1110 0010 0000 0110 ... 1110 1111 1111 1111	16	-	-

7.5.4.2 Select and Challenge commands

7.5.4.2.1 Manchester *Select* command

The Manchester version of the *Select* command inserts the 8 bit Short Interrogator ID field just before the CRC16. This field shall be included in the CRC calculation.

Table 7.15 — Manchester *Select* command

	CMD ID	Target	Action	Mem Bank	Pointer	Length	Mask	Truncate	Short Interrogator ID	CRC 16
# of bits	4	3	3	2	EBV	8	Variable	1	8	16
Description	1010	000: S0 001: S1 010: S2 011: S3 100: **SL** 101: RFU 110: RFU 111: RFU	See Table 6.30	00: RFU 01: UII 10: TID 11: User	Starting Mask address	Mask Length	Mask Value	0: Disable 1: Enable	Tag checks if Interrogator locking is in effect.	

Tags shall not reply to a *Select* command.

7.5.4.2.2 Manchester *Challenge* command

The Manchester version of the *Challenge* command inserts the 8 bit Short Interrogator ID field just before the CRC16. This field shall be included in the CRC calculation.

Table 7.16 — Manchester *Challenge* command

	CMD ID	RFU	IncRepLen	Immed	CSI	Length	Message	Short Interrogator ID	CRC 16
# of bits	8	2	1	1	8	8	Variable	8	16
Description	11010100	00	0: Omit length from reply 1: Include length in reply	0: Do not transmit result with EPC 1: Transmit result with EPC	CSI	length of message	message (depends on CSI	Tag checks if Interrogator locking is in effect.	

Tags shall not reply to a *Challenge* command.

7.5.4.3 Inventory commands

7.5.4.3.1 Manchester *Query_Bat*

7.5.4.3.1.1 Manchester *Query_Bat* command

Tags and Interrogators shall implement the Manchester *Query_Bat* command shown in Table 7.17 and Table 7.18. *Query_BAT* initiates and specifies an inventory round for battery assisted Tags.

The basic function of the "Tag Type Select Field" field is for selecting the categories of Tags to be included in the interrogation round without requiring a separate *Select* command. These categories include the current Battery Assisted Passive design, Tags featuring various kinds of sensors and sensor conditions (such as a Sensor Alarm that may need expedited handling), and RFU bits for future expansion. A Tag shall ignore the RFU bits within the Tag Type Select regardless of their values and only participate in a singulation round based on the currently defined selection bits.

The Tag shall send the reverse link backscatter signal on subcarrier frequency selected by the BLF field. The maximum required BLF shall be 640 kHz.

The field SS Response enables or disables Simple Sensor response in a Tag implementing Simple Sensor data for a given interrogation round. For Tags that do not support Simple Sensor data, the Tag shall ignore the value of SS Response.

Table 7.17 — Manchester *Query_BAT* command (see NOTES 1 to 4)

	CMD	QueryTag Type Select field	M	TRext	Sel	Ses-sion	Tar-get	Q	SS Resp	BLF See NOTE 2	T2ext	CW power Delta over forward power	Short Interro-gator ID	CRC-5
# of bits	8	8	4	1	2	2	1	4	1	4	1	2	8	5
Descrip-tion	1100 1100	NOTE 1	0000: M=1 0001: M=2 0010: M=4 0011: M=8 0100: M=16 0101: M=32 0110: M=64 0111: M=128 1000: M=256 1001 to 1111: RFU	0: No pilot tone 1: Use pilot tone	00: All 01: All 10:~**SL** 11: **SL**	00: S0 01: S1 10: S2 11: S3	0: A 1: B	0-15	0: No 1: Yes	Required values 0000: 25.2632 kHz 0001: 40 kHz 0010: 48 kHz 0011: 64 kHz 0100: 80 kHz 0101: 96 kHz 0110: 120 kHz 0111: 160 kHz 1000: 192 kHz 1001: 240 kHz 1010: 320 kHz 1011: 384 kHz 1100: 480 kHz 1101: 640 kHz Optional values: 1110: 960 kHz 1111: 1920 kHz	0: max 4Tbit 1: INACT_T or Global Timeout	11: +30 dB 10: +20 dB 01: +10 dB 00: 0 dB NOTE 3 NOTE 4	Tag checks if Interro-gator locking is in effect.	

NOTE 1 See Table 7.18 for Tag Type Select field options. Tags that match the chosen criteria will enter the Query rounds, while others will remain in the **battery ready** state. If Session Locking is in effect, then there must be a match between the Session specified in the *Query_BAT* command and the activating session for the Tag to respond to this command. See 7.5.3.4.2.2.

NOTE 2 Required BLF accuracy is 4% for BLF <= 640 kHz, and 1.5% for BLF > 640 kHz, over the temperature range that the Tag supports.

NOTE 3 This field indicates optional offset between Interrogator forward link transmit peak power and reverse backscatter supporting carrier power. The field shows that reverse link caririer power may be equal to or greater than forward Interrogator power in steps of 10 dB. Lower Interrogator forward power limits system interference (particularly Interrogator on Interrogator), while higher reverse link supporting carrier power relieves the reverse link limit of RFID systems with highly sensitive BAP Tags. See Annex Q for further information.

NOTE 4 Tolerance on the variable Interrogator transmit power levels is not specified, but is recommended as +/- 4 dB.

Table 7.18 — Manchester *Query_BAT* Tag Type Select field (NOTES 1 to 5)

Interpretation (NOTE 1)	**Sensor Alarm**	**Full Function Sensor**	**Simple Sensor**	**RFU**	**RFU**	**RFU**	**Battery Assisted Passive** (NOTE 4)
1	1	1	1	1	1	1	1
0: Inclusive 1: Exclusive	0: No 1: Yes	0: No 1: Yes	0: No 1: Yes	0	0	0	0: No 1: Yes

NOTE 1 This bit dynamically switches the interpretation of the remainder of the Tag Type Select field between "Inclusive" (the Tag responds if it matches any selection marked "Yes") and "Exclusive" (the Tag responds only if it matches all selected criteria marked "Yes").

NOTE 2 In this table, "Yes" means that category of Tag is considered for inclusion in the Query round. In the "Inclusive" case for a criteria listed as Yes the Tag will respond if it meets other criteria in the *Query* command. In the "Exclusive" case the Tag responds only if the Tag meets all criteria marked Yes as well as other Query criteria. Logically the Tag responds to "Inclusive" if a logical OR function of all marked Yes criteria = 1. The Tag responds to "Exclusive" if a logical AND of all marked Yes criteria = 1.

NOTE 3 RFU bits shall be set to zero until defined. If the Interpretation field is "0, Inclusive" the Tag shall respond if it matches at least one selected criteria (the Tag ignores the RFUs because the implied logical OR function is still a one). If the Interpretation field is "1, Exclusive", the Tag shall NOT respond if any RFU bits are non-zero (in that case the Tag does not know if it matches the implied AND function).

NOTE 4 This field means the Tag only supports a Manchester receiver and a backscatter transmitter.

NOTE 5 The Manchester *Query_BAT* Tag Type Select field is identical to the Manchester activation Tag Type Select Field.

7.5.4.3.1.2 Tag response to a Manchester *Query_BAT* command

A Tag that generates a slot counter value of 0 replies to Manchester *Query_BAT* command with an RN16. This response is similar to the Query response in Clause 6 except for the modifications to BLF, M, and frequency accuracy.

Table 7.19 — Tag response to a Manchester *Query_BAT* command

	Response
# of bits	16
Description	RN16

7.5.4.3.2 Manchester *QueryAdjust*

7.5.4.3.2.1 Manchester *QueryAdjust* command

The Manchester version of the *QueryAdjust* command inserts the 8 bit Short Interrogator ID field just before the CRC16. This field shall be included in the CRC calculation. Unlike the PIE *QueryAdjust*, which steps the slot counter parameter Q up and down, the Manchester version can immediately reset the Q to any allowed value.

Table 7.20 — Manchester *QueryAdjust* command

	CMD ID	Session	Q Value	Short Interrogator ID	CRC-16
# of bits	4	2	4	8	16
Description	1001	00: S0 01: S1 10: S2 11: S3	Q value from 0 to 15	Tag checks if Interrogator locking is in effect.	

7.5.4.3.2.2 Tag response to a Manchester *QueryAdjust* command

A Tag that generated a slot counter value of 0 replies to Manchester *QueryAdjust* command with RN16. This response is similar to the Query response in Clause 6 except for the modifications to BLF, M, and frequency accuracy.

Table 7.21 — Tag response to a Manchester *QueryAdjust* command

	RN16
# of bits	16
Description	RN16

7.5.4.3.3 Manchester *QueryRep*

7.5.4.3.3.1 Manchester *QueryRep* command

The Manchester version of the *QueryRep* command appends the 8 bit Short Interrogator ID to the end of the command only if Interrogator locking is in effect as indicated in the *Activation* command.

Table 7.22 — Manchester *QueryRep* command

	CMD ID	Session	Short Interrogator ID
# of bits	2	2	8
Description	00	00: S0 01: S1 10: S2 11: S3	Included only if Interrogator locking is in effect.

7.5.4.3.3.2 Tag response to a Manchester *QueryRep* command

A Tag that generated a slot counter value of 0 replies to Manchester *QueryRep* command with RN16. This response is similar to the Query response in Clause 6 except for the modifications to BLF, M, and frequency accuracy.

Table 7.23 — Tag response to a Manchester *QueryRep* command

	RN16
# of bits	16
Description	RN16

7.5.4.3.4 Manchester *ACK*

The Manchester *ACK* command has a modified behaviour from the equivalent command in clause 6.3.2.12.2.4. The modification makes Tags with greater sensitivity immune to *ACK* commands intended for other Tags sent by enemy Interrogators (see Annex B and Annex C for details). It works by preventing the change in present state caused by successfully decoding an *ACK* command with invalid RN16 or RN16_handle.

7.5.4.3.4.1 Manchester *ACK* command

The Manchester *ACK* command structure is shown in Table 7.24.

Table 7.24 — Manchester *ACK* command

	Command	RN
# of bits	2	16
Description	01	Echoed RN16 or handle

Table 7.25 — Tag reply to a successful *ACK* command

	Response
# of bits	See **C.3.4**
Description	See **C.3.4**

7.5.4.3.5 Manchester *NAK*

The Manchester *NAK* command includes two parameters, session, and short Interrogator ID, though the Interrogator ID is only transmitted if Interrogator locking is in effect. These two parameters provide different levels of interference rejection. When Interrogator locking is not in effect, the session parameter is used to avoid accidental return to arbitrate state when the session parameter does not match. A higher degree of protection is achieved by using Interrogator locking. Any Tag that successfully decodes a *NAK* command with the correct parameters shall transition to arbitrate state (except while in **battery ready** or **killed** states). Otherwise it shall ignore the command. When Interrogator locking is not in effect, this command shall be executed if session matching is observed and ignored otherwise. When Interrogator locking is in effect, this command shall be executed only if session and Interrogator ID match, otherwise, it shall be ignored.

7.5.4.3.5.1 Manchester *NAK* command

The Manchester *NAK* command structure is shown in Table 7.26.

Table 7.26 — Manchester *NAK* command

	CMD ID	Session	Short Interrogator ID
# of bits	8	2	8
Description	11000000	00: S0 01: S1 10: S2 11: S3	Included only if Interrogator locking is in effect.

Tags shall not reply to a *NAK* command.

7.5.4.3.6 Manchester *Next*

The *Next* command allows an Interrogator to send a single Tag to **hibernate** immediately after singulation or access with a single command-response interchange using the same RN16 from the singulation or access operation. The Tag response to a successful *Next* command gives positive confirmation that the Tag received the command to go to **hibernate**. Alternatively, an Interrogator can send Tags to **hibernate** using the *Deactivate_BAT* command on any and all Tags that match the selected flag conditions without confirmation responses.

7.5.4.3.6.1 Manchester *Next* command

The Manchester *Next* command moves an individual Tag back to **hibernate** using an RN16 or RN16_handle. This command is valid only in the acknowledged, open, and secured states.

Upon deactivation and return to the **hibernate** state, all flags without active timeouts in operation shall return to state A. Flags with active timeouts associated with other sessions that are in state *B* shall continue their timeouts, and upon timeout shall reset to state A. If a timeout was specified in the last activation, then the flag associated with the current Activation Session shall begin its timer operation with Session flag set to state B, and shall revert to state *A* upon timeout. If Session Locking is not in effect then the Tag returns to **hibernate** with all **inventoried** flags set to A. In all cases the Tag returns to **hibernate** with **SL** deasserted.

Tags may have been activated with one sensitivity setting and be taken back to **hibernate** with the same or different sensitivity setting as set by the Hibernate Sensitivity flag in the Manchester *Next* command.

Table 7.27 — Manchester *Next* command

	CMD ID	**Hibernate Sensitivity** **NOTE 1**	**RN**
# of bits	8	1	16
Description	1100 1011	0: low sensitivity (approx -30 dBm to +10 dBm) 1: high sensitivity (approximately -30 dBm and below)	RN16 or RN16_handle

NOTE Support for high sensitivity is optional.

7.5.4.3.6.2 Tag response to a Manchester *Next* command

The Tag response to a *Next* shall be as shown in Table 7.28. Once the Tag sends the *Next* response, it shall transition to the **hibernate** (full or stateful) state.

Table 7.28 — Tag response to *Next* command

	RN16
# of bits	16
Description	RN16 or RN16_handle

7.5.4.3.7 Manchester Deactivate_BAT

7.5.4.3.7.1 Manchester *Deactivate_BAT* command

The *Deactivate_BAT* command is used to conserve power in the Tag by taking groups of Tags no longer required for interrogation and moving them to their lowest power mode **hibernate** state. Tags and Interrogators shall implement the *Deactivate_BAT* command in Table 7.29. Tags shall execute a *Deactivate_BAT* command from any state (except killed).

In order to reduce the incidence of an Interrogator accidentally sending Tags that are in session with other Interrogators back to the **hibernate** state, the *Deactivate_BAT* command is selective regarding activation session (it obeys Session Locking when in effect) and also specific to that **inventoried** flag state. But, it also possesses Don't Cares also for both the **SL** and Session **inventoried** flag states when more broad application is desired. It also possesses an "Override" field for system resets that will send all Tags that receive the command back to **hibernate** regardless of any flag states.

Tags may have been activated with one sensitivity setting and be taken back to **hibernate** with the same or different sensitivity setting (if supported) as set by the Hibernate Sensitivity flag in the Manchester *Deactivate_BAT* command.

The Tag shall react to *Deactivate_BAT* as follows:

1. With Session Locking is in effect, then if Activation Session matches command supplied session, and if **SL** state and **inventoried** flag state (target) criteria both match (or if they are Don't Care), then the Tag goes to the **hibernate** or **stateful hibernate** state as a function of timer programming. All flags without active timeouts pending or in operation shall set **inventoried** flags to state A. Flags with active timeouts shall upon return to **hibernate** be set to state *B* with timers beginning (for last Activating Session) or continuing (for other timers that were in effect).

2. With Session Locking in effect and with the Activation Session matching the session specified in the command, but either or both of **SL** and session **inventoried** flag state not matching, the Tag stays in Normal Mode and goes to the **battery ready** state. Assuming well controlled system operation (where **inventoried** flag state *A* means not inventoried and *B* means inventoried), this allows for inventoried Tags to be sent back to **hibernate** while keeping non-inventoried Tags awake for a new Query round.

3. With Session Locking in effect but with the Activation Session not matching the session specified in the command, the Tag ignores the command. This deals with the case where a different Interrogator is properly targeting its own Tags, and a Tag it did not wake up has a timer running on that flag and thus has state *B* on that flag, but should not respond to that different Interrogator.

4. With Session Locking not in effect (and **hibernate** timers thus not used), the Tag shall respond to full match on **SL** and command indicated session **inventoried** flag state (or commanded Don't Care on these states) by returning to **hibernate** with all **inventoried** flags in state *A* and timers cleared.

5. With Session Locking not in effect, the Tag shall respond to mismatch on **SL** and match on command indicated session **inventoried** flag state by going to the **battery ready** state with no change in flag states. With Session Locking not in effect, with any state of **SL** match but mismatch on **inventoried** flag state, the Tag shall go to **battery ready** with **inventoried** flag set to A.

6. If the Override bit is set in the *Deactivate_BAT* and Interrogator Locking is not in effect, then this clears all timers and the Tag shall return to **hibernate** with all flags in state *A* (regardless of Session Locking, **inventoried** match, or **SL** match).

7. If Interrogator Locking is in effect, then Override clears all timers and Select if and only if there is an Interrogator match. Override takes precedence over Session Locking but not over Interrogator Locking.

This action shall elicit no response in any Tag.

Table 7.29 — Manchester *Deactivate_BAT* command

	CMD ID	Sel	Session	Target	inventoried flag use	Override	Hibernate Sensitivity	Interrogator ID	CRC
# of bits	8	2	2	1	1	1	1	8	16
Description	1100 1010	00: All 01: All 10: ~**SL** 11: **SL**	00:S0 01:S1 10:S2 11:S3	0: A 1: B	0: Don't care for **inventoried** state 1: Do care for **inventoried** state	0:Use **SL** and **inventoried** criteria 1:Go to **hibernate** Always (i.e. all sessions)	0: low sensitivity (approx -30 dBm to +10 dBm) 1: high sensitivity (approximately -30 dBm and below)	Tag checks if Interrogator locking is in effect.	

Table 7.30 provides examples to take into account the effects of Interrogator Locking.

Table 7.30 — *Manchester Deactivate_BAT* command procedures

Case	Interrogator Locking in effect	Interrogator match	Override	Session Locking in effect	Activating session match	Indicated Target matching (or commanded Don't Care)	SL State matching (or commanded Don't Care)	Action	Next state
1	Yes	No	X[2]	X	X	X	X	None	Current state
2	Yes	Yes	Yes	X	X	X	X	Clear Timers, set flags to A	**hibernate**
3	Yes	Yes	No	Yes	Yes	Yes	Yes	Session flag→B	**stateful hibernate**[1]
4	Yes	Yes	No	Yes	Yes	Yes	No	None	**battery ready**
5	Yes	Yes	No	Yes	Yes	No	X	Session flag→A	**battery ready**
6	Yes	Yes	No	Yes	No	X	X	None	Current state
7	No	X	Yes	X	X	X	X	Clear Timers, set flags to A	**hibernate**
8	No	X	No	Yes	Yes	Yes	Yes	Session flag→B	**stateful hibernate**[1]
9	No	X	No	Yes	Yes	Yes	No	None	**battery ready**
10	No	X	No	Yes	Yes	No	X	Session flag→A	**battery ready**
11	No	X	No	Yes	No	X	X	None	Current state

Table 7.30 *(continued)*

Case	Interrogator Locking in effect	Interrogator match	Override	Session Locking in effect	Activating session match	Indicated Target matching (or commanded Don't Care)	SL State matching (or commanded Don't Care)	Action	Next state
12	No	X	No	No	X	Yes	Yes	Session flag→B	**stateful hibernate**[1]
13	No	X	No	No	X	Yes	No	None	**battery ready**
14	No	X	No	No	X	No	X	Session flag→A	**battery ready**

NOTE 1 If timer was programmed during activation, Session flag→B while timer is operating, after timeout Session flag→A.

NOTE 2 X means that the parameter value does not matter (Don't Care).

7.5.4.3.8 Manchester Multirate_Reset

Multirate_Reset is added to provide an override command that can allow Interrogators (mostly portable, but this is not a requirement) to assert control over all awake Manchester Tags no matter what their current state machine state (except killed), **inventoried** flag value, and Forward Data Rate. The *Multirate_Reset* command takes all Tags back to **hibernate** (with specified **hibernate** sensitivity level), so that they may then be reactivated on a single Forward Data Rate. **inventoried** flags are set to *A* and **SL** flag to deasserted, and all associated timers are cleared. This provides a mechanism to start from a known fresh point.

7.5.4.3.8.1 Manchester *Multirate_Reset* command

This command is transmitted sequentially at all Interrogator supported data rates. In between data rates, a time T_4 of CW is allowed for Interrogators to switch data rates, and for Tags to end each command normally. The command code is presented in Table 7.31. A high level depiction of the command structure is given in Figure 7.36 (all possible data rates shown, but the Interrogator sends this command only at the data rates supported and in use in a local environment).

Table 7.31 — Manchester *Multirate_Reset* command

	CMD ID	Hibernate Sensitivity	CRC-16
# of bits	8	1	16
Description	1100 1110	0: low sensitivity (approx -30 dBm to +10 dBm) 1: high sensitivity (approximately -30 dBm and below)	

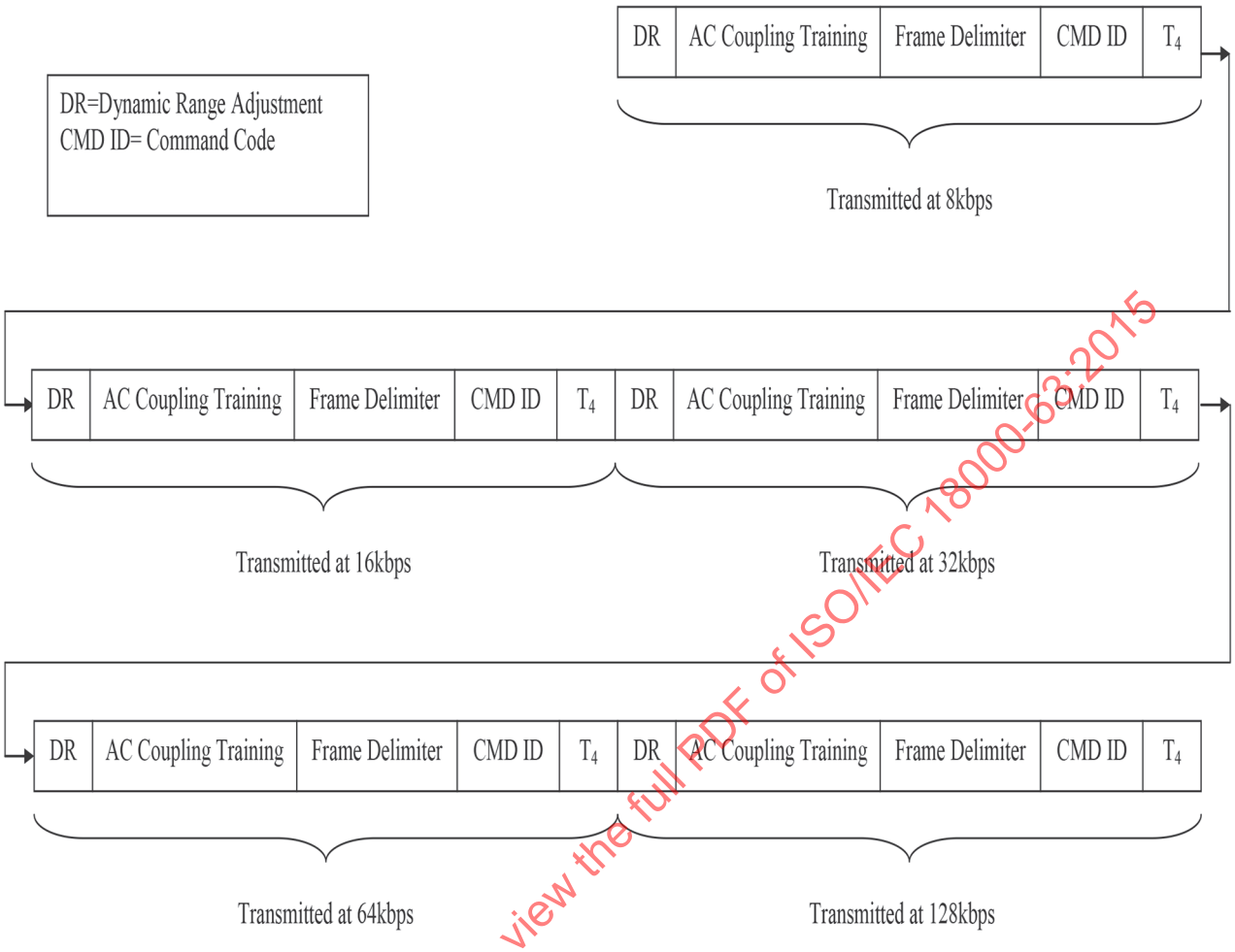


Figure 7.36 — *Multirate_Reset* command structure

The Tag shall not transmit any response to a *Multirate_Reset* command, but it does take the actions of clearing all **inventoried** flag timers, setting the **inventoried** flag states to A, setting the **SL** to deasserted (~**SL**), and then entering the full **hibernate** state.

7.5.4.4 Access commands

7.5.4.4.1 OpRegister Read/Write

Interrogators and Tags shall implement the *OpRegister Read/Write*. Only Tags in the secured state shall execute the *OpRegister Read/Write* command.

7.5.4.4.1.1 OpRegister Read/Write command

The *OpRegister Read/Write* command enables reading and writing specific configuration parameters that will be used in low level protocol operations by the Tag circuitry. This enables changing these configuration parameters with immediate application. The registers are not stored in the 4 general memory banks and cannot be read or written with the standard *Read* and *Write* commands, and cannot be matched using the *Select* command.

After issuing an *OpRegister Read/Write* command, an Interrogator shall transmit CW for the lesser of T_{REPLY} or 20 ms, where T_{REPLY} is the time between the Interrogator's *OpRegister Read/Write* command and the Tag's backscattered reply. An Interrogator may observe several possible outcomes from an *OpRegister Read/Write*, depending on the success or failure of the Tag's memory read/write operation.

Upon receiving a valid *OpRegister Read/Write* command with the R/W bit set to 1 indicating a write, a Tag shall write the commanded data into the addressed operational register. The Tag's reply in this write mode shall use the extended preamble, i.e. a Tag shall reply as if TRext=1 regardless of the TRext value in the *Query_BAT* that initiated the round.

Upon receiving a valid *OpRegister Read/Write* command with the R/W bit set to 0 indicating a read, the Tag's reply shall use the preamble indicated in the TRext value in the *Query_BAT* that initiated the round.

Table 7.32 — *OpRegister Read/Write command*

	Command	R/W	Reg ID	WordCount	Data	RN	CRC-16
# of bits	8	1	3	4	Variable	16	16
Description	1101 0000	0: Read 1: Write	000: Activation Code (primary) 001: Optional AC (secondary) 010:-111: RFU	Number of words to read/write 0001: MML only 0111: MML+6 AC word	Data to be written	handle	

For an OpRegister Read, if the WordCount is 0, the Tag shall backscatter the entire length of the register identified by Reg ID. For an OpRegister Write, if the WordCount is 0, the Tag shall ignore the command.

For the two AC words, if an invalid count of greater than 7 is received, or if a Reg ID with an RFU value is received, the Tag shall send an error code as defined in Annex K with a value of 00h indicated "other error." The lengths of the other OpRegister parameters are as defined within this standard.

7.5.4.4.1.2 Tag Response to an *OpRegister Read/Write* command

After successfully completing an *OpRegister Read/Write*, the Tag shall backscatter the reply shown in Table 7.33 within 20 ms of the command. If the Interrogator observes this reply within 20 ms, then the command completed successfully.

If the Tag encounters an error, the Tag shall backscatter an error code during the CW period rather than the reply shown in Table 7.33 (see Annex I for error-code definitions and for the reply format).

Table 7.33 — Tag reply to a successful *OpRegister Read/Write* command

	Header	Data	RN	CRC-16
# of bits	1	Read: Variable Write: 0	16	16
Description	0: No error	Data if read; none if write	handle	

7.5.4.5 System management commands

7.5.4.5.1 Manchester Broadcast ID

7.5.4.5.1.1 Broadcast ID command

The *Broadcast ID* command transmits the Interrogator ID and relevant parameters to assist with Interrogator system deployment and management. There are no Tag requirements associated with this command. Specialized tools capable of reading and decoding the Manchester *Broadcast ID* can use this command to perform RF measurements and associate other Interrogator commands containing only short Interrogator ID to a more complete long Interrogator ID. The instances in which the *Broadcast ID* command is used is determined by the implementer.

Table 7.34 — Manchester *Broadcast ID* command

	CMD ID	ID Length (words)	Long Interrogator ID	Short Interrogator ID	Antenna	Power	Channel	CRC-16
# of bits	8	3	Variable	8	8	8	13	16
Description	1100 1101	Length of long Interrogator ID (16-bit words)	Long Interrogator ID	System assigned short Interrogator ID	Antenna number	2's complement -64 to +63.5 dBm in 0.5 dB step	Channel number in 25 kHz increments; Ch 0 = 830.0 MHz	

7.5.4.5.1.2 Tag Response to a Manchester *Broadcast ID* command

The Tag shall not respond to a *Broadcast ID* command.

7.5.4.6 BAP Manchester Tag PIE support requirements

Manchester Tags and Interrogators shall also support PIE using the definition provided in sub-clause 7.2.

7.6 Extended Protocol Control

The optional Extended Protocol (XPC) Field has been defined to enable the Interrogator to support either sensors or more advanced battery operation or both. The first XPC word XPC_W1 provides additional information about Tag capabilities. To indicate to the Interrogator that XPC_W1 is supported, normal Protocol Control Bit 6 XI flag (the 7th bit of the PC word, which is UII bit 16_h) shall be used to indicate the presence of XPC_W1. The XPC_W1 shall consist of 16 additional protocol control bits as shown in Figure 7.37 and described in the below text and in Table 7.35.

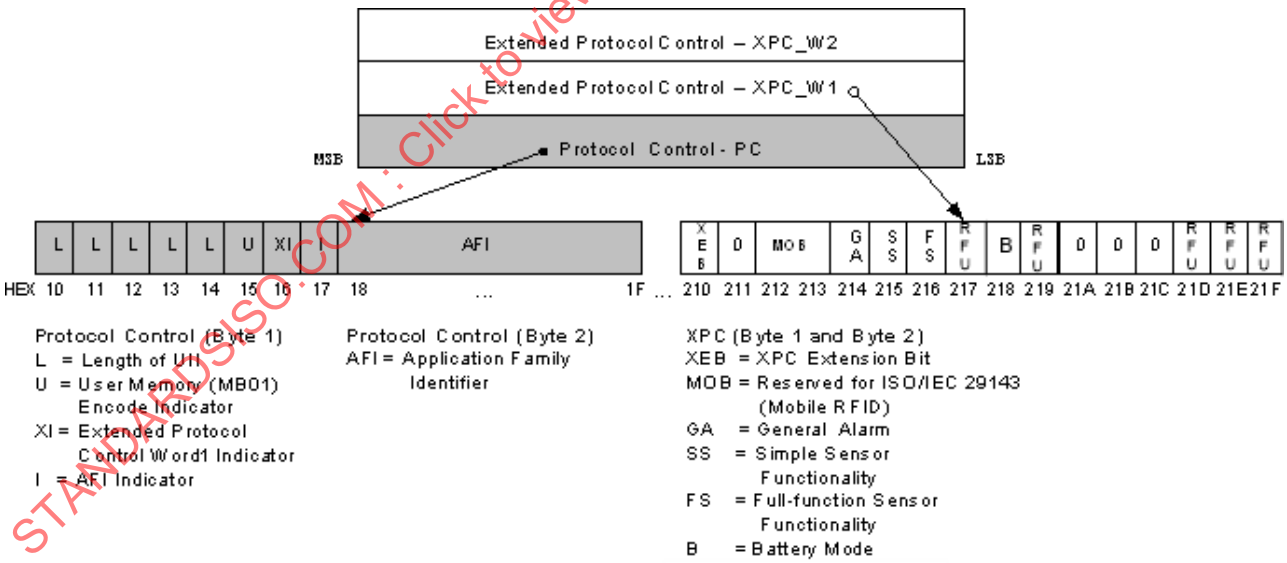


Figure 7.37 — Extended Protocol Control bit definitions (see NOTES 1 and 2)

NOTE 1 Figure 7.37 shows how Protocol Control and Extended Protocol Control are linked to each other and how the part of a Tag reply to an *ACK* command dedicated to PC and XPC is structured, assumed that both XPC_W1 and XPC_W2 (currently RFU) are supported. PC and XPC are not stored on subsequent addresses of the UII memory (see 6.3.2.1.2).

NOTE 2 At the option of the Tag manufacturer, the General Alarm (GA) bit may indicate one or more of sensor alarms, low battery, or any other exception conditions on the tag.

As defined in clause 6.3.2.1, the XPC_W1 begins in Bit 210_h and ends in BIT $21F_h$ in the UII memory. Bit 210h shall be the XPC Extension Bit (XEB). If a Tag does not implement an XPC_W2 then the XEB shall be zero. Bit 214_h shall be the Sensor Alarm (SA) flag, Bit 215_h shall be the Simple Sensor (SS) flag, and Bit 216_h shall be the Full Function Sensor (FS) flag. Bit 218_h shall be set according to the supported mode as defined in Table 7.35

All remaining bits of XPC_W1 as well as all bits of XPC_W2 are currently reserved for future use (RFU) and shall be set to 0 at default.

The Sensor Alarm bit shall be set to 1 to indicate the violation of at least one alarm condition in at least one of the attached sensors.

The detailed meaning of combinations of the XPC_W1 flags is described in Table 7.35 and in its detailed notes.

Table 7.35 — Meaning of combinations of B field, and SS and FS bits in XPC

B Field Bits 218_h	SS Bit Bit 215_h	FS Bit Bit 216_h	Meaning
0	0	0	Passive Tag without sensors.
0	0	1	Passive Tag with at least one Full-function Sensor but not Simple Sensor.
0	1	0	Passive Tag with one pre-configured Simple Sensor.
0	1	1	Passive Tag with at least one Full-function Sensor and one pre-configured Simple Sensor, or Simple Sensor Functionality supported by one Full-function sensor.
1	0	0	BAP Tag without sensors. If TC = 0.
1	0	1	BAP Tag and at least one Full-function Sensor.
1	1	0	BAP Tag with one pre-configured Simple Sensor.
1	1	1	BAP Tag, at least one Full-function Sensor available and one pre-configured Simple Sensor available, or Simple Sensor Functionality supported by one Full-function sensor.

8 Sensor support

8.1 Applicability

In case an Interrogator or Tag supports any command, response or feature of Clause 8 then this Interrogator or Tag shall support all mandatory commands, responses or features and it may support all optional commands, responses or features of Clause 8.

In case an Interrogator or Tag does not support any command, response or feature of Clause 8 then Clause 8 does not apply for this device.

8.2 Overview

This clause describes an optional extension to Type C that adds sensor support. However, references to RFID air interface commands, variable names, and state names in this clause refer to Type C unless otherwise noted. An RFID Tag having sensor support must have a Real Time Clock (RTC) to support sensor operations. The RTC and its use are defined in 8.3.

Two classes of sensor are supported:

— **Simple Sensor:** A Simple Sensor is programmed at source and is not required to be user programmed. A Simple Sensor by its nature delivers its payload as a Simple Sensor Data (SSD) block, appended to

the object-related UII if requested by the Interrogator (see 8.5) as a Tag is inventoried, and therefore does not require a separate dialogue to collect the sensor data.

The three fundamental characteristics of a Simple Sensor are:

— the sensor data is appended to the UII during Tag arbitration if requested by the Interrogator,

— the sensor does not need to be user programmed and

— the sensor computes pass/fail based on its characteristics, however it may also provide more details (e.g. an 8 bit sensor value)

The SSD block includes information about:

— the type of sensor, limited to a few basic environmental features

— measurement spans

— thresholds

— alarm status, indicating pass / fail conditions

The SSD block is read by any user and provides limited configuration capabilities of the Simple Sensor by authorized (password controlled) user only. See Annex O and Annex P for details.

Simple Sensors are defined in 8.5.

— **Full Function Sensor:** Full Function Sensors provide greater flexibility than Simple Sensors, by:

— supporting a greater variety of sensor types and measurement spans,

— enabling thresholds to be set within a wider range

— capturing and processing different types of data

The Full Function Sensor senses and records a set of measurements, stores the measurements and delivers them to an Interrogator when instructed to do so. Access to the sensor data requires the Tag to first be singulated and then a dialogue conducted between the Tag and an Interrogator. A Full Function Sensor may if required be programmed by the user either once or on multiple occasions.

Full Function Sensors are defined in 8.6.

Tags may be equipped with one or more sensors. If a Type C Tag has sensor support then bit 16_h (XI) of the UII memory shall be asserted and XPC_W1 shall be supported, see also 6.3.2.1.2.

If it has the capability, an RFID Tag may support a combination of Full Function Sensors and one Simple Sensors. Each sensor shall be fully compliant with the class of sensor.

8.3 Real Time Clock (RTC)

8.3.1 General

RFID Tags having sensor support shall have a 32-bit RTC with an LSB of 1 second and the RTC shall be used as the source of UTC timestamps for sensor related data. Timestamps shall be based on the UTC time epoch beginning at 1970-01-01 00:00:00. At configuration of a sensor, the RTC shall be set to the current 32-bit UTC time precise to 1 second.

8.3.2 Setting the RTC

The RTC shall be accessible via the RTC Address. The RTC Address shall be stored in the TID memory (memory bank 10_2) at memory word 28_h MSB first (see Figure 8.38). The RTC Address shall comprise 6

bits reserved for future use (RFU) followed by 2 bits identifying the memory bank (MB) where the RTC is stored, and a 24-bit address in non EBV format specifying the starting word of the RTC.

Table 8.1 — Structure of RTC Address

	RFU	MB	Word Address
# of bits	6	2	24
description	Reserved for future use	Memory bank selector	RTC starting word address

The RTC may be set using the *Write* or *BlockWrite* commands. Write operations may be restricted by the use of write locking with access password, write permalocking, or other means implemented by the RFID Tag. It shall not be permitted to write a UTC time of zero into the RTC nor set the RTC when an alarm condition is present other than the low battery alarm.

The current 32-bit UTC timestamp shall be transmitted by the Interrogator to set the RTC during sensor configuration and at other times to maintain time synchronisation.

The 32-bit UTC timestamp may be taken from the Interrogator if it supports UTC time as specified in this clause. Timestamps that are not in seconds and of a 32-bit format are not suitable. Timestamps that are based on local time are not acceptable and shall not be used.

If the Interrogator is not capable of providing a timestamp in the specified format, then this should be provided using some accurate network-based UTC time. Some services compliant with IEEE 1588 can deliver a 64-bit timestamp, where the 32 MSB bits identify time to a second. Services compliant with the internet Simple Network Time Protocol, as defined in IETF RFC 1769, might also be able to provide the time as a 32-bit value that can be directly used.

The current 32-bit UTC timestamp within the RTC may be obtained using the *Read* command.

8.3.3 BroadcastSync command (optional)

Interrogators and Tags may implement the *BroadcastSync* command shown in Table 8.2. *BroadcastSync* allows an Interrogator to provide current UTC time information to a population of Tags to allow for synchronisation of the RTC time within a Tag to that of the clock time within the Interrogator. Time shall be a 32-bit integer value representing UTC time with an LSB of one second. It shall not be permitted to write a UTC time of zero into the RTC nor set the RTC when an alarm condition is present other than the low battery alarm.

The *BroadcastSync* command includes a CRC-16 that is calculated over the first command-code bit to the last UTC Time bit. If a Tag receives a *BroadcastSync* with a valid CRC-16 it may update its internal time to that of the UTC Time. If the RTC provides sufficient accuracy then it may decide to ignore the *BroadcastSync* command. The *BroadcastSync* command may update the RTC regardless if the RTC is located within a memory region that is locked or permalocked. An Interrogator may issue a *BroadcastSync* command at any time although it is recommended to use the command prior to the start of an inventory session. There shall be no reply from a Tag in response to a *BroadcastSync* command and the Tag shall not make any state transitions as a result of a *BroadcastSync* command.

Table 8.2 — *BroadcastSync* command (optional)

	Command	UTC Time	CRC-16
# of bits	8	32	16
description	1101 0001	Current Time	

8.3.4 Time synchronisation

The RTC mechanism on an RFID Tag might not be able to maintain an accurate record of time. This is partly a factor of the cost of providing additional accuracy and the impact on temperature variation to which the RTC is exposed. If the RFID Tag implements sensor event records, then it shall also implement

a time synchronisation record block to enable an application to reconstruct a more accurate time line. The information about this record is given in TID memory words $2D_h$, $2E_h$, and $2F_h$.

The number of 32-bit timestamps that are required to be stored on the RFID Tag may be kept to a minimum by using the ordinal number of the sensor sample to reduce the use of memory and air interface transmission.

The time synchronisation record block enables a UTC timestamp to be recorded against the most recent sample count value. Each recorded time synchronisation record shall comprise, in sequence:

— The prevailing time as recorded by the RFID Tag. This may be a 32-bit RTC value maintained since the time of configuration (Record Type is set to 0 in Figure 8.38), or the 16-bit sample count value (Record Type is set to 1 in Figure 8.38). The 32-bit structure is recommended where time accuracy is important or where there is a large sample interval.

— The 32-bit UTC timestamp.

The number of time synchronisation records contained within the time synchronisation record block shall be specified in the Number of Records in word $2F_h$ in TID (see Figure 8.38). Every time the Tag decides to write a time synchronisation record, the Number of Records field is updated accordingly. The timestamp accuracy at the application level shall be achieved provided that a sufficient time synchronisation update rate is maintained. The time synchronisation update rate should define a maximum time interval between updates necessary to maintain timestamp accuracy as well as a minimum time interval between updates necessary to generate a time synchronisation record. When the allocated memory for the time synchronisation block becomes full no further records are possible, but no error shall be indicated.

8.3.4.1 Reconciling UTC with the 32-bit RTC time

As both the RTC and UTC timestamps have the same common datum, the UTC timestamp at configuration, it is possible to determine the extent of the difference in time at each point when time synchronisation took place. The extent of any difference might vary over the sample period and because of different causes, the RTC could be 'fast' or 'slow' without any pattern. Any time correction can only be applied more accurately close to the time of synchronisation, and be less certain at a point midway between two time synchronisation events.

8.3.4.2 Reconciling UTC with the 16-bit sample count

In this simpler approach, the best level of synchronisation that can be achieved is to write the UTC timestamp against the most recent sample count value. This means that time synchronisation is also a factor of the size of the sample interval. The larger the sample interval, the more likely that the UTC timestamp will represent a time that is between the most recent sample count value and the next sample count value. This provides a simple indication:

— that the RTC and UTC are within the same time period as determined by the sample interval, or

— if the UTC indicates an earlier time than the sample count, then the RTC is 'fast', or

— if the UTC indicates a later time than sample count + 1, then the RTC is 'slow'.

It is only when the synchronisation is at these outer limits that any inaccuracy can be corrected.

8.4 HandleSensor command (optional)

To provide the means to support a broad range of different sensors types and hence different command sets, the *HandleSensor* command provides a transport mechanism to carry commands for intelligent sensors as a payload. The related response to the Interrogator shall carry the response of the sensor after receiving and processing the specific sensor command.

Tags shall interpret the payload of the *HandleSensor* command and execute the according sensor access command locally or forward it to its sensors for processing if required and transmit the response to the command within a well defined response time.

The *HandleSensor* command includes a 7 bit PortNr field which can be used to forward the included payload to a specific sensor without further inspection of the encapsulated sensor command. PortNr logically addresses a specific sensor attached to a Tag. Logical sensor addresses are assigned in ascending order starting at 0_2, e.g. 0_2, 1_2, 10_2, and so on.

The structure of *HandleSensor* shall be as shown in Table 8.3.

Tags execute *HandleSensor* only from the open or secured states. The CRC-16 is calculated over the first command code bit to the last handle bit.

An Interrogator may observe several possible outcomes from a *HandleSensor* command:

The *HandleSensor* command succeeds: After completing the *HandleSensor* command a Tag shall backscatter the reply shown in Table 4 comprising a header (a 0-bit), the optional sensor response to the encapsulated sensor command (applies only if the bit Response Expected has been set to value 1_2 in the *HandleSensor* command), the Tag's handle, and a CRC-16 calculated over the 0-bit to the last handle bit. The reply to *HandleSensor* shall begin within 20 ms.

The Tag encounters an error: The Tag shall backscatter an error code (see Annex I for error-code definitions and for the reply format).

Table 8.3 — Structure of *HandleSensor* command

	Command	PortNr	Payload Size	Payload	Response Expected	Response Length	RN	CRC-16
# of bits	8	7	Variable	Variable	1	Variable	16	16
description	11011001	Logical sensor address	Length of the Pay-load in bits (EBV-8)	Sensor command	1 = true 0 = false	Expected length of the sensor response in bits (EBV-8)	handle	

Table 8.4 — Tag reply to successful *HandleSensor* command

	Header	Response (optional)	RN	CRC-16
# of bits	1	Variable	16	16
Description	0	Sensor response	handle	

8.5 Simple Sensor

Although called "Simple Sensors", the devices are required to support features common to any type of sensor device. The Simple Sensor has to monitor the environmental characteristic for which it is designed, take samples at defined intervals, compare and process against criteria, and report its status.

The Simple Sensor transmits its data as a Simple Sensor Data (SSD) block which is appended to the Tag UII if requested by the Interrogator. The *Flex_Query* command in clause 7.4.1 and *Query_BAT* command in 7.5.4.3.1 contain the field SSD Resp to authorize Tags to append the SSD block after an *ACK* command. The SSD block shall consist of 32 bits or 48 bits depending on the type of Simple Sensor.

The specification of Simple Sensors also includes the provisions of the Sensor Directory System (SDS) as specified in sub-clause 8.6.

8.5.1 Type C and Simple Sensor

Simple sensors shall be implemented either as a memory mapped Simple Sensor or as a ported Simple Sensor. A memory mapped Simple Sensor is accessed via memory read and write operations to the RFID Tag. A ported Simple Sensor is accessed via dedicated sensor commands to the RFID Tag encapsulated in the *HandleSensor* transport command defined in 8.4.

The Simple Sensor Data (SSD) shall be accessible via the SSD Address. The SSD Address shall be stored in the TID memory (memory bank 10_2) at memory word 26_h MSB first (see Figure 8.38). The MSB of the SSD Address shall indicate the access method for the Simple Sensor. If the MSB contains a logical 0, then the access method is memory mapped and the remainder of the SSD Address is comprised of 3 bits reserved for future use (RFU), followed by 2 bits identifying the SSD size, followed by 2 bits identifying the memory bank (MB) where the SSD is stored, and a 24-bit address in non EBV format specifying the starting word of the SSD block. If the MSB contains a logical 1, then the access method is ported and the remainder of the SSD Address is comprised of 7 bits for a port number and 24 bits reserved for future use.

Table 8.5 — Structure of SSD Address for Memory Mapped Simple Sensor

	Access Method	RFU	SSD Size	MB	Word Address
# of bits	1	3	2	2	24
Description	'0' Memory Mapped	Reserved for future use	00 = 32 bits 01 = 48 bits 10 = RFU 11 = RFU	Memory bank selector	SSD starting word address

Table 8.6 — Structure of SSD Address for Ported Simple Sensor

	Access Method	PortNr	RFU
# of bits	1	7	24
Description	'1' Ported	Port Number	Reserved for future use

NOTE 1 The SSD Address is not necessarily required to be factory programmed and can be changed if the Tag has more than one sensor and it is decided to support another sensor from a certain time onwards.

NOTE 2 For Simple Sensors the SSD Address as well as the SSD itself may be write locked to the Interrogator, though alarms remain Tag/sensor writable.

NOTE 3 Tags supporting a Simple Sensor may indicate the necessity to request SSD to the Interrogator by using the Sensor Alarm flag (214h) of the XPC, see 6.3.2.1.2.5.

NOTE 4 Products developed before this document is officially published may transmit SSD at default without the requirement of explicitly requesting the Simple Sensor Data block.

NOTE 5 For Tags having more than one Simple Sensor, there is a mechanism provided to select the Active Simple Sensor, see 8.6.1.3.

Tags that support a Simple Sensor shall use the extended protocol control described in 6.3.2.1.2.5 to make this detail visible to the Interrogator. Within XPC_W1 the Simple Sensor (SS) bit 215_h shall be set to 1 to indicate the presence of a Simple Sensor, 0 else.

The SSD block comprises the sensor type, measure span, ACC, sampling regime, high in-range limit, low in-range limit, monitor delay, high out-of-range alarm delay, low out-of-range alarm delay, and alarms, all as defined in Annex O.

The contents of the Simple Sensor Data Block, see Annex O, shall be transmitted by the Tag subsequent to the UII after receiving an *ACK* in the course of an inventory round in which transmission of Simple Sensor Data is enabled by properly setting the SSD Resp field in *Flex_Query* or *Query_BAT*. SSD shall not

be transmitted as default. The SSD block transmission as requested by the Interrogator by asserting the appropriate flag in the inventory command (see 8.5) is defined in Table 8.7.

The reply to the *ACK* command in case Simple Sensor Data transmission is requested is defined in Table 8.7. In general, Simple Sensor Data is appearing like an extension of the XPC data, which is the reason why it is never mentioned as explicit data in other clauses, however, only XPC is mentioned.

If Simple Sensor Data is transmitted as part of the reply to the *ACK* command, a dynamic PacketPC shall be used instead of the StoredPC available at UII memory location 10_h – 14_h, where the 5 most significant bits of the PacketPC included in the Tag response shall be used to indicate the length of the UII+XPC, but not extended by the SSD as shown in Table 8.7. In this context length always refers to 16-bit words.

Table 8.7 — Tag reply to a successful *ACK* command if SSD is requested

	Response	SSD	PacketCRC
# of bits	21 to 480 or 464 (see NOTE 1)	32 or 48	16
Description	PC word, XPC words, UII	Simple Sensor Data (according to Annex O)	CRC-16

NOTE 1 If a Tag does not support XPC_W2 then the maximum of UII + SSD is 480 bits. If a Tag supports XPC_W2 then the maximum length of UII + SSD to be backscattered is reduced by one word to accommodate the optional XPC_W2, so is 464 bits.

PacketCRC is a dynamic CRC to be calculated over all bits of the response to an *ACK* command received prior to the CRC-16.

8.5.1.1 Memory mapped Simple Sensor

A memory mapped Simple Sensor is accessed by an Interrogator using memory read and write operations to the RFID Tag.

The Simple Sensor may be configured using the *Write* or *BlockWrite* commands and writing the entire Simple Sensor Data block as defined in Annex O into the RFID Tag memory. Write operations to the Simple Sensor Data block may be restricted by the use of write locking with access password, write permalocking, or other means implemented by the RFID Tag. The RFID Tag may be implemented such that some parameter fields (e.g. sensor type, span, accuracy) are not writeable by an Interrogator and these bits shall be ignored by the RFID Tag when processing the *Write* or *BlockWrite* commands. The RFID Tag shall not permit setting alarm conditions when processing the *Write* or *BlockWrite* commands.

The Simple Sensor Data block may be obtained using the *Read* command.

Configuring the memory mapped Simple Sensor shall reset all the alarms within the Simple Sensor Data block, stop the RTC, and initialize the RTC to zero. The memory mapped Simple Sensor is disarmed when the RTC has a zero value. Writing the current time to the RTC shall then start the RTC running and arm the memory mapped Simple Sensor. Once armed, the sensor shall be monitored as configured and check for the event conditions required to declare an alarm condition. If an alarm condition other than low battery is declared, then the RTC shall be stopped and its current time retained as a timestamp for when the event occurred.

8.5.1.2 Ported Simple Sensor

A ported Simple Sensor is accessed by an Interrogator using dedicated sensor commands to the RFID Tag. The sensor commands are delivered to the Tag as the payload encapsulated within the *HandleSensor* command defined in 8.4. The ported Simple Sensor command set is defined in Annex P.

A number of records are defined for inclusion on a mandatory or optional basis to support the processing, encoding and subsequent diagnostics of a ported Simple Sensor. The ported Simple Sensor record structures are defined in Annex P.

8.6 Sensor Directory System and Full Function Sensors

The Sensor Directory System (SDS) is a simple directory system to assist in efficient accessing of sensors and sensor data. It applies to all sensor types, including Simple Sensors of both the memory mapped and ported sub-types, and to Full Function Sensors.

Full functions sensors may optionally be implemented on an 18000-63 Type C Tag. The Full Function Sensor characteristics and capabilities are given in the IEEE 1451.7 standard. Within XPC_W1 the Full Function Sensor (FS) bit 216_h shall be set to 1 to indicate the presence of a Full Function Sensor. These sensors are ported through a logical port. Sensor memory management is abstracted through the use of sensor records that are matched to dedicated sensor commands. These commands are transported to the sensor through the *HandleSensor* command defined in 8.4. The Full Function Sensor setup (available sensors, ports, types, etc) of a Tag is specified in the Sensor Directory System (SDS) in regular Tag memory.

8.6.1 Sensor Access – General Approach

Sensor access is handled via a fixed address for an initial hook to sensor information and mapped memory positions via SDS. All the information needed to access a particular sensor is given its SDS Entry.

8.6.1.1 SDS Address

Tags equipped with one or more Full Function Sensors shall provide a 32-bit SDS Address pointing to the starting word address of a Sensor Directory System. The SDS Address shall be stored in the TID memory (memory bank 10_2) at memory word 22_h MSB first.

It shall comprise of 6 bits reserved for future use (RFU), followed by 2 bits identifying the memory bank (MB) where the SDS is stored, and a 24-bit field specifying the starting word address of the SDS in linear (non EBV) format.

Table 8.8 — Structure of SDS Address

	RFU	MB	Word Address
# of bits	6	2	24
Description	Reserved for future use	Memory bank selector	SDS starting word address

The default value for the SDS Address shall be 0 for no sensor. Tags with one or more Full Function Sensors shall have a SDS Address ≠ 0.

Figure 8.38 shows a detailed layout of a Tag's TID memory if sensor access via SDS and Simple Sensor are supported.

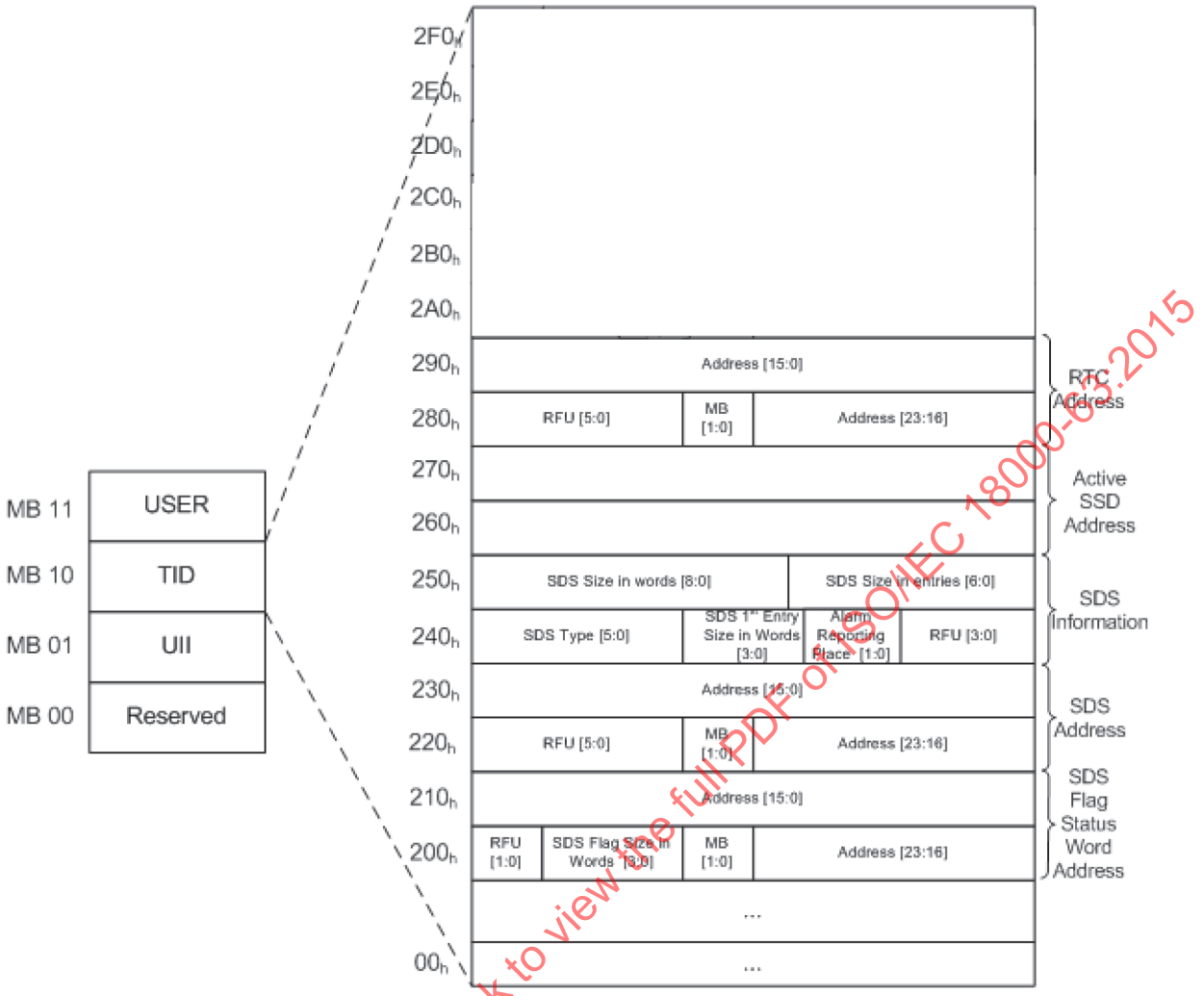


Figure 8.38 — Structure of TID memory

8.6.1.2 Sensor Directory System (SDS) pointers

The Sensor Directory System (SDS) is pointed to by two pointers located in TID, namely, the SDS Flag Status Word Address, and the SDS address, additionally, the SDS Information words are provided to access the SDS more efficiently. The structure of those pointers and the SDS Information words is given in Figure 8.38.

The SDS Flag Status Word Address is provided so that sensor alarms may be given in a separate memory location from the SDS Entries, in this way, the SDS entries can be completely hardcoded. The 24-bit address is in non EBV format. The SDS Flag Status Word is followed by one or more SDS Flag words.

Table 8.9 — SDS Flag Status Word structure

Word	Structure			
1	RFU [5:0]	Alarm Flag Latency [3:0]	BlockPermalock Status [1:0]	Lock Status [3:0]

The interpretation of the Alarm Flag Latency is given in Table 8.10.

Table 8.10 — Alarm Flag Latency field interpretation

Code	Meaning
0000	Sensor Alarms not supported
0001	Interrupt driven (near zero latency)
0010	Duty cycle driven
0011	Activation driven
0100	Poll driven
0101	Duty cycle, activation, and poll driven
0110	Sensor sample rate driven
0111 - 1111	RFU

The BlockPermalock field description is given in Table 8.11. This refers to the BlockPermalock status of the SDS Flag Status word and the SDS Flag words.

Table 8.11 — BlockPermalock Field interpretation

Code	Meaning
00	Not reported
01	Not BlockPermalocked
10	BlockPermalocked
11	RFU

The Lock Status field description is given in Table 8.12. This refers to the Lock status if the SDS Flag Status word and the SDS Flag words.

Table 8.12 — Lock Status Field interpretation

Code	Meaning
0000	Temp writable from **open** and **secured** states (Lock action 00)
0001	Permanently writable from **open** and **secured** states (Lock action 01)
0010	Temp writable from **secured** but not from **open** (Lock action 10)
0011	Temp not writable from any state (Lock action 11)
0100-1110	RFU
1111	Lock status is not reported

The structure of the SDS Flag word(s) is given in Figure 8.39.

Word	Structure															
1	SDS Entry 16 Alarm	.	.	.	.	.	.	.	.	.	.	.	SDS Entry 4 Alarm	SDS Entry 3 Alarm	SDS Entry 2 Alarm	SDS Entry 1 Alarm

Figure 8.39 — SDS Flag word(s) structure

More SDS Flag words are added as needed, and the number of words is given in the SDS Flag address word.

The SDS Information field makes the access to the SDS simpler and more efficient. It provides the SDS Type to enable future definitions; the only SDS Type defined in this standard is 000000_2 all other values are RFU. The First Entry Size in words is provided so that an Interrogator can read the SDS entries one

by one, or with the aid of the SDS Size in words, read the whole SDS. The Alarm Reporting Place field interpretation is given in the Table 8.13.

Table 8.13 — Alarm Reporting Place field interpretation

Code	Meaning
00	Alarms Not reported
01	Alarms Reported in SDS Flag word only
10	Alarms Reported in SDS Entry only
11	Alarms Reported in SDS Flag word and SDS Entry

8.6.1.3 SDS Entry Record

The SDS Entry Record Structure is shown in Figure 8.40.

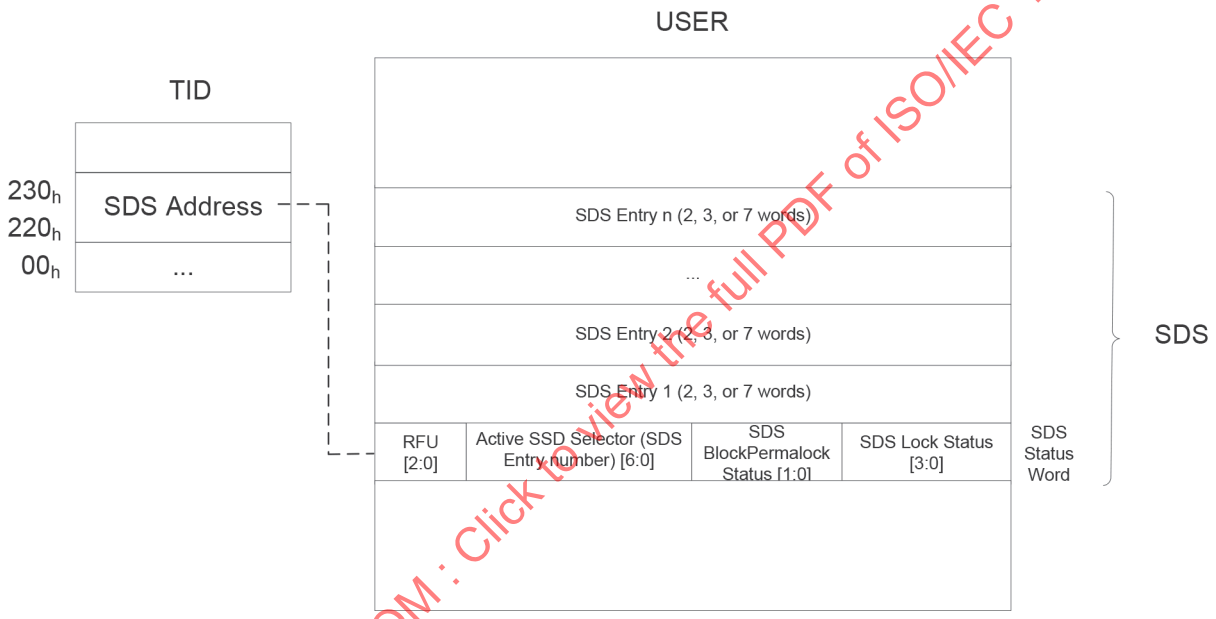


Figure 8.40 — SDS

The first word of the SDS is a status word. The structure of the SDS Status word is given in Figure 8.40. Within the SDS Status Word of Fig. 87, the SDS BlockPermalock Status field bits are as given in Table 8.11, and the SDS Lock Status field bits are as given in Table 8.12. For a Tag that has multiple Simple Sensor capabilities, writing the SDS entry into the "Active SSD Selector" field selects the SSD block that is backscattered after an *ACK* command (if so authorized by the Interrogator). This also routes the SSD block to the SSD address (if memory mapped) or updates the port number of the Simple Sensor (if ported). Since an SDS entry may correspond to a Simple Sensor or a Full Function Sensor, the Tag must backscatter an error code when the "Active SSD Selector" field is written with a Full Function Sensor Entry number. Additionally, the Tag must update the SSD size field accordingly.

Each sensor (Simple of Full Function) must have an entry in the SDS (with the exception of Tags with only one Simple Sensor since the information needed to access it is given in the TID pointers see Figure 8.38).

8.6.1.3.1 SDS Entry for a Memory Mapped Simple Sensor

The structure for this kind of entry is shown in Table 8.14.

Table 8.14 — SDS Entry for Memory Mapped Simple Sensors (3 words always)

Words	Structure				
1 (Lowest Memory Location)	Sensor Access Method [1:0]	Standard Identifier [5:0]	Sensor Type [3:0] (Annex O)	Sensor Alarm [0]	Next Entry Size in Words [2:0]
2	RFU [5:0]	MB [1:0]	SSD Address [23:16]		
3 (Highest Memory Location)	SSD Address [15:0]				

The interpretation of the Sensor Access Method is given in Table 8.15.

Table 8.15 — Sensor Access Method Interpretation

Code	Meaning
00	Simple Sensor Memory Mapped
01	Simple Sensor Ported
10	RFU
11	Full Function Sensor Ported

The interpretation of the Standard Identifier field is given in Table 8.16.

Table 8.16 — Standard Identifier Interpretation

Code	Meaning
000000	RFU
000001	IEEE 1451.7
000010	ISO/IEC 18000 63 Annex O
000011	ISO/IEC 18000 63 Annex P
000100~111111	RFU

Depending on the choice of the manufacturer, the Sensor Alarm field may or may not contain the alarm information (see Table 8.13). The 24-bit SSD address is not in EBV format.

8.6.1.3.2 SDS Entry for a Ported Simple Sensor

The structure for this kind of entry is shown in Table 8.17.

Table 8.17 — SDS Entry for Ported Simple Sensors (2 words always)

Words	Structure				
1 (Lowest Memory Location)	Sensor Access Type [1:0]	Standard Identifier [5:0]	Sensor Type [3:0] Annex O	Sensor Alarm [0]	Next Entry Size in Words [2:0]
2 (Highest Memory Location)	Port Number [6:0]	RFU [8:0]			

Port Number logically addresses a specific sensor attached to a Tag. Logical sensor addresses are assigned in ascending order beginning with 0000000. The number of sensors attached to a Tag can be determined from the number of SDS Entries field in the SDS information words.

8.6.1.3.3 SDS Entry for a Ported Full Function Sensor

The structure for this kind of entry is shown in Table 8.18.

Table 8.18 — SDS Entry for Full Function Sensors (3 or 7 words)

Words	Structure							
1	Sensor Access Type [1:0]	Standard Identifier [5:0]		Sensor Type [6:0] (See NOTE 1)				Sensor Alarm [0]
2	Next Entry Size in Words [2:0]	Port Number [6:0]				TEDS Type [2:0] (see NOTE 1)	AI Security Function Code [2:0] (see NOTE 1)	
3	AI Security Indicator [0] (see NOTE 1)	Sensor Security Indicator [1:0] (see NOTE 1)	Sensor Security Function Code [2:0] (see NOTE 1)	Authentication Encryption Function Code [2:0] (see NOTE 1)		5 bits Units Extension [4:0] (see NOTE 1)		RFU [1:0]
4 (optional)	Sensor ID [63:48] (see NOTE 1)							
5 (optional)	Sensor ID [47:32] (see NOTE 1)							
6 (optional)	Sensor ID [31:16] (see NOTE 1)							
7 (optional)	Sensor ID [15:0] (see NOTE 1)							

NOTE These fields come from the IEEE 1451.7 standard.

Annex A
(normative)

Extensible bit vectors (EBV)

An *extensible bit vector* (EBV) is a data structure with an extensible data range.

An EBV is an array of blocks. Each block contains a single extension bit followed by a specific number of data bits. If B represents the total number of bits in one block, then a block contains B – 1 data bits. Although a general EBV may contain blocks of varying lengths, Tags and Interrogators manufactured according to this protocol shall use blocks of length 8 bits (EBV-8).

The data value represented by an EBV is simply the bit string formed by the data bits as read from left-to-right, ignoring the extension bits.

Tags and Interrogators shall use the EBV-8 word format specified in Table A.1.

Table A.1 — EBV-8 word format

	0	0	0000000				
	1	0	0000001				
$2^7 - 1$	127	0	1111111				
2^7	128	1	0000001	0	0000000		
$2^{14} - 1$	16383	1	1111111	0	1111111		
2^{14}	16384	1	0000001	1	0000000	0	0000000

Because each block has 7 available data bits, an EBV-8 can represent numeric values between 0 and 127 with a single block. To represent the value 128, set the extension bit to 1 in the first block, and append a second block to the EBV-8. In this manner, an EBV-8 can represent arbitrarily large data values.

This protocol uses EBV-8 values to represent memory addresses and mask lengths.

Annex B
(normative)

State-transition tables

State-transition tables B.1 to B.1.7 shall define a Tag's response to Interrogator commands for passive PIE tags. For BAP PIE: see B.2 and for BAP Manchester: see B.3

The term "handle" used in the state-transition tables is defined in 6.3.2.6.5; error codes are defined in Table I.2; "slot" is the slot-counter output shown in Figure 6.21 and detailed in Annex J; and "–" in the "Action" column means that a Tag neither executes the command nor backscatters a reply.

B.1 State transition tables for passive

B.1.1 Present state: Ready

Table B.1 — Ready state-transition table

Command	Condition	Action	Next State
Query [1]	slot=0; matching **inventoried** & **SL** flags	backscatter new RN16	reply
	slot<>0; matching **inventoried** & **SL** flags	–	arbitrate
	otherwise	–	ready
QueryRep	all	–	ready
QueryAdjust	all	–	ready
ACK	all	–	ready
NAK	all	–	ready
Req_RN	all	–	ready
Select	correct parameters	assert or deassert **SL**, or set **inventoried** to *A* or *B*	ready
	incorrect parameters	–	ready
Read	all	–	ready
Write	all	–	ready
Kill	all	–	ready
Lock	all	–	ready
Access	all	–	ready
BlockWrite	all	–	ready
BlockErase	all	–	ready
BlockPermalock	all	–	ready
Challenge	supported security timeout, unsupported CSI, not-executable message, nonzero RFU bits	set **C**=0	ready
	supported CSI & executable message	store result, set **C**=1	ready
Authenticate	all	–	ready
AuthComm	all	–	ready

Table B.1 *(continued)*

Command	Condition	Action	Next State
SecureComm	all	–	ready
ReadBuffer	all	–	ready
KeyUpdate	all	–	ready
Untraceable	All	–	ready
FileSetup	All	–	ready
FileOpen	All	–	ready
FilePrivilege	All	–	ready
TagPrivilege	All	–	ready
FileList	All	–	ready
Flex_Query [1]	slot=0; matching **Tag Type Select** criteria, **inventoried** & **SL** flags	backscatter new RN16	reply
	slot<>0; matching **Tag Type Select** criteria, **inventoried** & **SL** flags	–	arbitrate
	Otherwise	–	ready
BroadcastSync	All	–	ready
HandleSensor	All	–	ready
Faulty	invalid [2]	–	ready

1: *Query* and *Flex_Query* start a new round and may change the session. *Query* and *Flex_Query* also instruct a Tag to load a new random value into its slot counter.

2: "Invalid" shall mean a command not recognizable by the Tag such as (1) an erroneous command (example: a command with an incorrect length field), (2) a command with a CRC error, or (3) an unsupported command.

B.1.2 Present state: Arbitrate

Table B.2 — Arbitrate state-transition table

Command	Condition	Action	Next State
Query [1,2]	slot=0; matching **inventoried** & **SL** flags	backscatter new RN16	reply
	slot<>0; matching **inventoried** & **SL** flags	–	arbitrate
	otherwise	–	ready
QueryRep	session matches inventory round & slot=0 after decrementing slot counter	decrement slot counter; backscatter new RN16	reply
	session matches inventory round & slot<>0 after decrementing slot counter	decrement slot counter	arbitrate
	session does not match inventory round	–	arbitrate

Table B.2 *(continued)*

Command	Condition	Action	Next State
QueryAdjust [2]	session matches inventory round & slot=0	backscatter new RN16	reply
	session matches inventory round & slot<>0	–	arbitrate
	session does not match inventory round	–	arbitrate
ACK	all	–	arbitrate
NAK	all	–	arbitrate
Req_RN	all	–	arbitrate
Select	correct parameters	assert or deassert **SL**, or set **inventoried** to *A* or *B*	ready
	incorrect parameters	–	arbitrate
Read	all	–	arbitrate
Write	all	–	arbitrate
Kill	all	–	arbitrate
Lock	all	–	arbitrate
Access	all	–	arbitrate
BlockWrite	all	–	arbitrate
BlockErase	all	–	arbitrate
BlockPermalock	all	–	arbitrate
Challenge	supported security timeout, unsupported CSI, not-executable message, nonzero RFU bits	set **C**=0	ready
	supported CSI & executable message	store result, set **C**=1	ready
Authenticate	all	–	arbitrate
AuthComm	all	–	arbitrate
SecureComm	all	–	arbitrate
ReadBuffer	all	–	arbitrate
KeyUpdate	all	–	arbitrate
Untraceable	all	–	arbitrate
FileSetup	all	–	arbitrate
FileOpen	all	–	arbitrate
FilePrivilege	all	–	arbitrate
TagPrivilege	all	–	arbitrate
FileList	all	–	arbitrate
Flex_Query [1,2]	slot=0; matching **Tag Type Select** criteria, **inventoried** & **SL** flags	backscatter new RN16	reply
	slot<>0; matching **Tag Type Select** criteria, **inventoried** & **SL** flags	–	arbitrate
	otherwise	–	ready
BroadcastSync	all	–	arbitrate
HandleSensor	all	–	arbitrate
Faulty	invalid [3]	–	arbitrate

1: *Query* and *Flex_Query* start a new round and may change the session.

2: *Quer y, Flex_Query* and *QueryAdjust* instruct a Tag to load a new random value into its slot counter.

3: "Invalid" shall mean a command not recognizable by the Tag such as (1) an erroneous command (example: a command with an incorrect length field), (2) a command with a CRC error, or (3) an unsupported command.

B.1.3 Present state: Reply

Table B.3 — Reply state-transition table

Command	Condition	Action	Next State
Query [1,2]	slot=0; matching **inventoried** & **SL** flags	backscatter new RN16	reply
	slot<>0; matching **inventoried** & **SL** flags	–	arbitrate
	otherwise	–	ready
QueryRep	session matches inventory round	–	arbitrate
	session does not match inventory round	–	reply
QueryAdjust [2]	session matches inventory round & slot=0	backscatter new RN16	reply
	session matches inventory round & slot<>0	–	arbitrate
	session does not match inventory round	–	reply
ACK	correct RN16	See Table 6.17	acknowledged
	incorrect RN16	–	arbitrate
NAK	all	–	arbitrate
Req_RN	all	–	arbitrate
Select	correct parameters	assert or deassert **SL**, or set **inventoried** to *A* or *B*	ready
	incorrect parameters	–	reply
Read	all	–	arbitrate
Write	all	–	arbitrate
Kill	all	–	arbitrate
Lock	all	–	arbitrate
Access	all	–	arbitrate
BlockWrite	all	–	arbitrate
BlockErase	all	–	arbitrate
BlockPermalock	all	–	arbitrate
Challenge	supported security timeout, unsupported CSI, not-executable message, nonzero RFU bits	set **C**=0	ready
	supported CSI & executable message	store result, set **C**=1	ready
Authenticate	all	–	arbitrate
AuthComm	all	–	arbitrate
SecureComm	all	–	arbitrate

Table B.3 *(continued)*

Command	Condition	Action	Next State
ReadBuffer	all	–	arbitrate
KeyUpdate	all	–	arbitrate
Untraceable	all	–	arbitrate
FileSetup	all	–	arbitrate
FileOpen	all	–	arbitrate
FilePrivilege	all	–	arbitrate
TagPrivilege	all	–	arbitrate
Flex_Query [1,2]	slot=0; matching **Tag Type Select** criteria, **inventoried** & **SL** flags	backscatter new RN16	reply
	slot<>0; matching **Tag Type Select** criteria, **inventoried** & **SL** flags	–	arbitrate
	otherwise	–	ready
BroadcastSync	all	–	reply
HandleSensor	all	–	arbitrate
FileList	all	–	arbitrate
T_2 timeout	See Figure 6.18 and Table 6.16	–	arbitrate
Faulty	invalid [3]	–	reply

1: *Query* and *Flex_Query* start a new round and may change the session.

2: *Query, Flex_Query* and *QueryAdjust* instruct a Tag to load a new random value into its slot counter.

3: "Invalid" shall mean a command not recognizable by the Tag such as (1) an erroneous command (example: a command with an incorrect length field), (2) a command with a CRC error, or (3) an unsupported command.

B.1.4 Present state: Acknowledged

Table B.4 — Acknowledged state-transition table

Command	Condition	Action	Next State
Query [1]	slot=0; matching **inventoried** [2] & **SL** flags	backscatter new RN16; transition inventoried [2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	reply
	slot<>0; matching **inventoried** [2] & **SL** flags	transition **inventoried** [2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	arbitrate
	otherwise	transition **inventoried** from *A*→*B* or *B*→*A* if and only if new session matches prior session	ready
QueryRep	session matches inventory round	transition **inventoried** from *A*→*B* or *B*→*A*	ready
	session does not match inventory round	–	acknowledged
QueryAdjust	session matches inventory round	transition **inventoried** from *A*→*B* or *B*→*A*	ready
	session does not match inventory round	–	acknowledged

Table B.4 *(continued)*

Command	Condition	Action	Next State
ACK	correct RN16	See Table 6.17	acknowledged
	incorrect RN16	–	arbitrate
NAK	all	–	arbitrate
Req_RN	correct RN16 & access password<>0	backscatter handle	open
	correct RN16 & access password=0	backscatter handle	secured
	incorrect RN16	–	acknowledged
Select	correct parameters	assert or deassert **SL**, or set **inventoried** to *A* or *B*	ready
	incorrect parameters	–	acknowledged
Read	all	–	arbitrate
Write	all	–	arbitrate
Kill	all	–	arbitrate
Lock	all	–	arbitrate
Access	all	–	arbitrate
BlockWrite	all	–	arbitrate
BlockErase	all	–	arbitrate
BlockPermalock	all	–	arbitrate
Challenge	supported security timeout, unsupported CSI, not-executable message, nonzero RFU bits	set **C**=0	ready
	supported CSI & executable message	store result, set **C**=1	ready
Authenticate	all	–	arbitrate
AuthComm	all	–	arbitrate
SecureComm	all	–	arbitrate
ReadBuffer	all	–	arbitrate
KeyUpdate	all	–	arbitrate
Untraceable	all	–	arbitrate
FileSetup	all	–	arbitrate
FileOpen	all	–	arbitrate
FilePrivilege	all	–	arbitrate
TagPrivilege	all	–	arbitrate
FileList	all	–	arbitrate
Flex_Query [1]	slot=0; matching **Tag Type Select** criteria, **inventoried** [2] & **SL** flags	backscatter new RN16; transition inventoried [2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	reply
	slot<>0; matching **Tag Type Select** criteria, **inventoried** [2] & **SL** flags	transition **inventoried** [2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	arbitrate
	otherwise	transition **inventoried** from *A*→*B* or *B*→*A* if and only if new session matches prior session	ready
BroadcastSync	all	–	acknowledged
HandleSensor	all	–	arbitrate

Table B.4 *(continued)*

Command	Condition	Action	Next State
T_2 timeout	See Figure 6.18 and Table 6.16	–	arbitrate
Faulty	invalid [3]	–	acknowledged

1: *Query* and *Flex_Query* start a new round and may change the session. *Query* and *Flex_Query* also instruct a Tag to load a new random value into its slot counter.

2: As described in 6.3.2.10, a Tag transitions its **inventoried** flag prior to evaluating the condition.

3: "Invalid" shall mean a command not recognizable by the Tag such as (1) an erroneous command (example: a command with an incorrect length field), (2) a command with a CRC error, or (3) an unsupported command.

B.1.5 Present state: Open

Table B.5 — Open state-transition table

Command	Condition	Action	Next State
Query [1]	slot=0; matching **inventoried** [2] & **SL** flags	backscatter new RN16; transition **inventoried** [2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	reply
	slot<>0; matching **inventoried** [2] & **SL** flags	transition **inventoried** [2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	arbitrate
	otherwise	transition **inventoried** from *A*→*B* or *B*→*A* if and only if new session matches prior session	ready
QueryRep	session matches inventory round	transition **inventoried** from *A*→*B* or *B*→*A*	ready
	session does not match inventory round	–	open
QueryAdjust	session matches inventory round	transition **inventoried** from *A*→*B* or *B*→*A*	ready
	session does not match inventory round	–	open
ACK	correct handle	See Table 6.17	open
	incorrect handle	–	arbitrate
NAK	all	–	arbitrate
Req_RN	all	backscatter new RN16	open
Select	correct parameters	assert or deassert **SL**, or set **inventoried** to *A* or *B*	ready
	incorrect parameters	–	open
Read	all	backscatter data	open
Write	all	backscatter header when done	open

Table B.5 *(continued)*

Command	Condition	Action	Next State
Kill (see also Figure 6.24)	supported security timeout	backscatter error code	open
	password-based kill & correct nonzero kill password	backscatter header when done	killed
	password-based kill & incorrect nonzero kill password	may set security timeout	arbitrate
	password-based kill & kill password=0	backscatter error code	open
	authenticated kill	backscatter error code; may set security timeout	arbitrate
Lock	all	–	open
Access (see also Figure 6.26)	supported security timeout	backscatter error code	open
	correct access password	backscatter handle	secured
	incorrect access password	may set security timeout	arbitrate
BlockWrite	all	backscatter header when done	open
BlockErase	all	backscatter header when done	open
BlockPermalock	all	–	open
Challenge	supported security timeout, unsupported CSI, not-executable message, nonzero RFU bits	set **C**=0	ready
	supported CSI and executable message	store result, set **C**=1	ready
Authenticate	supported security timeout	backscatter error code	open
	executable & senrep=0	store result; set **C**=1, backscatter response when done	open or secured [3]
	executable & senrep=1	backscatter response when done	open or secured [3]
	crypto error	see crypto suite	arbitrate
	new authentication	reset crypto engine	open
AuthComm	supported security timeout	backscatter error code	open
	prior Tag authentication & executable	backscatter response when done	see encapsulated command
	no prior Tag authentication	backscatter error code	open
	crypto error	see crypto suite	arbitrate
SecureComm	supported security timeout	backscatter error code	open
	prior Tag authentication, executable, & senrep=0	store result; set **C**=1, backscatter response when done	see encapsulated command
	prior Tag authentication, executable, & senrep=1	backscatter response when done	see encapsulated command
	no prior Tag authentication	backscatter error code	open
	crypto error	see crypto suite	arbitrate
ReadBuffer	**C**=1	backscatter data	open
	C=0	backscatter error code	open
KeyUpdate	all	–	open

Table B.5 *(continued)*

Command	Condition	Action	Next State
Untraceable	all	–	open
FileSetup	all	–	open
FileOpen	executable	close current file; open requested file; backscatter file info	open
FilePrivilege	all	–	open
TagPrivilege	all	–	Open
FileList	executable & senrep=0	store result; set **C**=1, backscatter response when done	Open
	executable & senrep=1	backscatter response when done	Open
Flex_Query [1]	slot=0; matching **Tag Type Select** criteria, **inventoried** [2] & **SL** flags	backscatter new RN16; transition **inventoried** [2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	Reply
	slot<>0; matching **Tag Type Select** criteria, **inventoried** [2] & **SL** flags	transition **inventoried** [2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	Arbitrate
	otherwise	transition **inventoried** from *A*→*B* or *B*→*A* if and only if new session matches prior session	Ready
BroadcastSync	all	–	Open
HandleSensor	all	backscatter data	Open
Faulty	unsupported parameters [4]	backscatter error code	Open
	incorrect handle [5]	none unless specified by crypto suite	Open
	improper [6]	–	Arbitrate
	invalid [7]	none unless specified by crypto suite	Open

1: *Query* and *Flex_Query* start a new round and may change the session. *Query* and *Flex_Query* also instruct a Tag to load a new random value into its slot counter.

2: As described in 6.3.2.10, a Tag transitions its **inventoried** flag prior to evaluating the condition.

3: See cryptographic suite.

4: "Unsupported parameters" shall mean an access command with a correct handle and CRC and that is recognizable by the Tag but contains or specifies (1) a nonzero or incorrect RFU value, (2) an unsupported CSI; (3) an encapsulated command that is unsupported or disallowed, (4) an unsupported or incorrect memory bank, memory location, address range, or FileNum, (5) a hidden or locked memory bank or location, (6) an unsupported file or files, (7) a command that requires encapsulation but is nonetheless unencapsulated (see Table 6.28), (8) a *delayed* or *in-process* reply and the specified operation causes the Tag to encounter an error, (9) an operation for which the Interrogator has insufficient privileges, (10) an unsupported cryptographic parameter, or (11) other parameters not supported by the Tag.

5: "Incorrect handle" shall mean an access command with a correct CRC and that is recognizable by the Tag but has an incorrect handle. The cryptographic suite indicated by CSI in the prior *Challenge* or *Authenticate* command may specify that a Tag reset its cryptographic engine upon receiving a security command with an incorrect handle.

6: "Improper" shall mean a command (except *Req_RN* or *Query*) that is recognizable by the Tag but is interspersed between successive *Kill* or *Access* commands in a password-based kill or access command sequence, respectively (see Figure 6.24 and Figure 6.26).

7: "Invalid" shall mean a command not recognizable by the Tag such as (1) an erroneous command (example: a command with an incorrect length field or a *BlockWrite*/*BlockErase* with a zero-valued WordCount), (2) a command with a CRC error, (3) an unsupported command, or (4) a *Write* command for which the immediately preceding command was not a *Req_RN*. The cryptographic suite indicated by CSI in the prior *Challenge* or *Authenticate* command may specify that a Tag reset its cryptographic engine upon receiving an invalid command.

B.1.6 Present state: Secured

Table B.6 — Secured state-transition table

Command	Condition	Action	Next State
Query [1]	slot=0; matching **inventoried** [2] & **SL** flags	backscatter new RN16; transition inventoried [2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	reply
	slot<>0; matching **inventoried** [2] & **SL** flags	transition **inventoried** [2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	arbitrate
	otherwise	transition **inventoried** from *A*→*B* or *B*→*A* if and only if new session matches prior session	ready
QueryRep	session matches inventory round	transition **inventoried** from *A*→*B* or *B*→*A*	ready
	session does not match inventory round	–	secured
QueryAdjust	session matches inventory round	transition **inventoried** from *A*→*B* or *B*→*A*	ready
	session does not match inventory round	–	secured
ACK	correct handle	See Table 6.17	secured
	incorrect handle	–	arbitrate
NAK	all	–	arbitrate
Req_RN	all	backscatter new RN16	secured
Select	correct parameters	assert or deassert **SL**, or set **inventoried** to *A* or *B*	ready
	incorrect parameters	–	secured
Read	all	backscatter data	secured
Write	all	backscatter header when done	secured
Kill (see also Figure 6.24)	supported security timeout	backscatter error code	secured
	password-based kill & correct nonzero kill password	backscatter header when done	killed
	password-based kill & incorrect nonzero kill password	may set security timeout	arbitrate
	password-based kill & kill password=0	backscatter error code	secured
	authenticated kill with prior Interrogator authentication & AuthKill privilege	backscatter response when done	killed
	authenticated kill but no prior Interrogator authentication or no AuthKill privilege	backscatter error code; may set security timeout	arbitrate

Table B.6 *(continued)*

Command	Condition	Action	Next State
Lock	all	backscatter header when done	secured
Access (see also Figure 6.26)	supported security timeout	backscatter error code	secured
	correct access password	backscatter handle	secured
	incorrect access password	may set security timeout	arbitrate
BlockWrite	all	backscatter header when done	secured
BlockErase	all	backscatter header when done	secured
BlockPermalock	Read/Lock=0	backscatter permalock bits	secured
	Read/Lock=1	backscatter header when done	secured
Challenge	supported security timeout, unsupported CSI, not-executable message, nonzero RFU bits	set **C**=0	ready
	supported CSI and executable message	store result, set **C**=1	ready
Authenticate	supported security timeout	backscatter error code	secured
	executable & senrep=0	store result; set **C**=1, backscatter response when done	secured
	executable & senrep=1	backscatter response when done	secured
	crypto error	see crypto suite	arbitrate
	new authentication	reset crypto engine	open
AuthComm	supported security timeout	backscatter error code	secured
	prior Interrogator authentication & executable	backscatter response when done	see encapsulated command
	no prior Interrogator authentication	backscatter error code	secured
	crypto error	see crypto suite	arbitrate
SecureComm	supported security timeout	backscatter error code	secured
	prior Interrogator authentication, executable, & senrep=0	store result; set **C**=1, backscatter response when done	see encapsulated command
	prior Interrogator authentication, executable, & senrep=1	backscatter response when done	see encapsulated command
	no prior Interrogator authentication	backscatter error code	secured
	crypto error	see crypto suite	arbitrate
ReadBuffer	**C**=1	backscatter data	secured
	C=0	backscatter error code	secured
KeyUpdate	supported security timeout	backscatter error code	secured
	prior Interrogator authentication, executable, & senrep=0	store result; set **C**=1, backscatter response when done	secured
	prior Interrogator authentication, executable, & senrep=1	backscatter response when done	secured
	no prior Interrogator authentication	backscatter error code	secured
	crypto error	see crypto suite	arbitrate
Untraceable	executable	backscatter header when done	secured

Table B.6 *(continued)*

Command	Condition	Action	Next State
FileSetup	executable & senrep=0	store result; set **C**=1, backscatter response when done	secured
	executable & senrep=1	backscatter response when done	secured
FileOpen	executable	close current file; open requested file; backscatter file info	secured
FilePrivilege	executable & senrep=0	store result; set **C**=1, backscatter response when done	secured
	executable & senrep=1	backscatter response when done	secured
TagPrivilege	executable & senrep=0	store result; set **C**=1, backscatter response when done	secured
	executable & senrep=1	backscatter response when done	secured
FileList	executable & senrep=0	store result; set **C**=1, backscatter response when done	secured
	executable & senrep=1	backscatter response when done	secured
Flex_Query [1]	slot=0; matching **Tag Type Select** criteria, **inventoried** [2] & **SL** flags	backscatter new RN16; transition inventoried [2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	reply
	slot<>0; matching **Tag Type Select** criteria, **inventoried** [2] & **SL** flags	transition **inventoried** [2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	arbitrate
	otherwise	transition **inventoried** from *A*→*B* or *B*→*A* if and only if new session matches prior session	ready
BroadcastSync	all	–	secured
HandleSensor	all	backscatter data	secured
Faulty	unsupported parameters [3]	backscatter error code	secured
	incorrect handle [4]	none unless specified by crypto suite	secured or open [4]
	improper [5]	–	arbitrate
	invalid [6]	none unless specified by crypto suite	secured or open [6]

1: *Query* and *Flex_Query* start a new round and may change the session. *Query* and *Flex_Query* also instruct a Tag to load a new random value into its slot counter.

2: As described in 6.3.2.10 a Tag transitions its inventoried flag prior to evaluating the condition.

3: "Unsupported parameters" shall mean an access command with a correct handle and CRC and that is recognizable by the Tag but contains or specifies (1) a nonzero or incorrect RFU value, (2) an unsupported CSI; (3) an encapsulated command that is unsupported or disallowed, (4) an unsupported or incorrect memory bank, memory location, address range, lock payload, blockpermalock payload, KeyID, or FileNum, (5) a hidden or locked memory bank or location, (6) an unsupported file or files, (7) insufficient or unallocateable memory, (8) an unencrypted message that requires encryption, (9) a command that requires encapsulation but is nonetheless unencapsulated (see Table 6.28), (10) a *delayed* or *in-process* reply and the specified operation causes the Tag to encounter an error, (11) an RFU privilege value, (12) an operation for which the Interrogator has insufficient privileges, (13) an unsupported cryptographic parameter, or (14) other parameters not supported by the Tag.

4: Incorrect handle" shall mean an access command with a correct CRC and that is recognizable by the Tag but has an incorrect handle. The default next state is **secured**, but the cryptographic suite indicated

by CSI in the prior *Challenge* or *Authenticate* command may specify that a Tag reset its crypto engine and transition to the **open** state upon receiving a security command with an incorrect handle.

5: "Improper" shall mean a command (except *Req_RN* or *Query*) that is recognizable by the Tag but is interspersed between successive *Kill* or *Access* commands in a password-based kill or access command sequence, respectively (see Figure 6.24 and Figure 6.26).

6: "Invalid" shall mean a command not recognizable by the Tag such as (1) an erroneous command (example: a command with an incorrect length field or a *BlockWrite*/*BlockErase* with a zero-valued WordCount), (2) a command with a CRC error, (3) an unsupported command, or (4) a *Write* command for which the immediately preceding command was not a *Req_RN*. The default next state is **secured**, but the cryptographic suite indicated by CSI in the prior *Challenge* or *Authenticate* command may specify that a Tag reset its cryptographic engine and transition to the **open** state upon receiving an invalid command.

B.1.7 Present state: Killed

Table B.7 — Killed state-transition table

Command	Condition	Action	Next State
Query	all	–	killed
QueryRep	all	–	killed
QueryAdjust	all	–	killed
ACK	all	–	killed
NAK	all	–	killed
Req_RN	all	–	killed
Select	all	–	killed
Read	all	–	killed
Write	all	–	killed
Kill	all	–	killed
Lock	all	–	killed
Access	all	–	killed
BlockWrite	all	–	killed
BlockErase	all	–	killed
BlockPermalock	all	–	killed
Challenge	all	–	killed
Authenticate	all	–	killed
AuthComm	all	–	killed
SecureComm	all	–	killed
ReadBuffer	all	–	killed
KeyUpdate	all	–	killed
Untraceable	all	–	killed
FileSetup	all	–	killed
FileOpen	all	–	killed
FilePrivilege	all	–	killed
TagPrivilege	all	–	killed
FileList	all	–	killed
Flex_Query	all	–	killed
BroadcastSync	all	–	killed

Table B.7 *(continued)*

Command	Condition	Action	Next State
HandleSensor	all	–	killed
Faulty	all	–	killed

B.2 State transition tables for BAP PIE

B.2.1 Present state: sleep

Table B.8 — sleep state-transition table

Command	Condition	Action	Next State
N/A	**Sleep** timer expires	–	listen or stateful listen
N/A	RF detected by optional **sleep** mode detector	–	battery ready or stateful battery ready

B.2.2 Present state: low power listen

Table B.9 — low power listen state-transition table

Command	Condition	Action	Next State
N/A	RF detected by **low power listen** mode detector	–	battery ready or stateful battery ready

B.2.3 Present state: listen or stateful listen

Table B.10 — listen or stateful listen state-transition table

Command	Condition	Action	Next State
N/A	RF detected within time LT	–	battery ready or stateful battery ready
N/A	No RF detected within time LT	–	stateful sleep or stateful low power listen

B.2.4 Present state: stateful sleep or stateful low power listen

Table B.11 — stateful sleep or stateful low power listen state-transition table

Command	Condition	Action	Next State
N/A	**sleep** timer expires	–	listen or stateful listen
N/A	Persistence timers all expire	–	sleep or low power listen
N/A	RF detected by optional **sleep** mode detector or **low power listen** mode detector	–	battery ready or stateful battery ready

B.2.5 Present state: battery ready

Table B.12 — battery ready state-transition table

Command	Condition	Action	Next State
Query [1]	slot=0; matching **inventoried** & **SL** flags	backscatter new RN16	reply
	slot<>0; matching **inventoried** & **SL** flags	–	arbitrate
	Otherwise	–	battery ready
QueryRep	All	–	battery ready
QueryAdjust	All	–	battery ready
ACK	All	–	battery ready
NAK	All	–	battery ready
Req_RN	All	–	battery ready
Select	correct parameters	assert or deassert **SL**, or set **inventoried** to *A* or *B*	battery ready
	incorrect parameters	–	battery ready
Read	All	–	battery ready
Write	All	–	battery ready
Kill	All	–	battery ready
Lock	All	–	battery ready
Access	All	–	battery ready
BlockWrite	All	–	battery ready
BlockErase	All	–	battery ready
BlockPermalock	All	–	battery ready
Challenge	supported security timeout, unsupported CSI, not-executable message, nonzero RFU bits	set **C**=0	battery ready
	supported CSI & executable message	store result, set **C**=1	battery ready
Authenticate	All	–	battery ready
AuthComm	All	–	battery ready
SecureComm	All	–	battery ready
ReadBuffer	All	–	battery ready
KeyUpdate	All	–	battery ready
Untraceable	All	–	battery ready
FileSetup	All	–	battery ready
FileOpen	All	–	battery ready
FilePrivilege	All	–	battery ready
TagPrivilege	All	–	battery ready
FileList	All	–	battery ready

Table B.12 *(continued)*

Command	Condition	Action	Next State
Flex_Query [1]	slot=0; matching **Tag Type Select** criteria, **inventoried** & **SL** flags	backscatter new RN16	reply
	slot<>0; matching **Tag Type Select** criteria, **inventoried** & **SL** flags	–	arbitrate
	otherwise	–	battery ready
BroadcastSync	All	–	battery ready
HandleSensor	All	–	battery ready
Faulty	invalid [2]	–	battery ready
INACT_T [3] or (Selective) Global Timeout	BAP Tag supports Battery Saver Mode	–	stateful sleep or stateful low power listen
	BAP Tag does not support Battery Saver Mode	–	battery ready or stateful battery ready

1: *Query* and *Flex_Query* start a new round and may change the session. *Query* and *Flex_Query* also instructs a Tag to load a new random value into its slot counter.

2: "Invalid" shall mean a command not recognizable by the Tag such as (1) an erroneous command (example: a command with an incorrect length field), (2) a command with a CRC error, or (3) an unsupported command.

3: Details about these timers can be found in sub-clauses 7.3.2.2 and 7.3.2.3.

B.2.6 Present state: Arbitrate

Table B.13 — Arbitrate state-transition table

Command	Condition	Action	Next State
Query [1,2]	slot=0; matching **inventoried** & **SL** flags	backscatter new RN16	reply
	slot<>0; matching **inventoried** & **SL** flags	–	arbitrate
	otherwise	–	battery ready
QueryRep	session matches inventory round & slot=0 after decrementing slot counter	decrement slot counter; backscatter new RN16	reply
	session matches inventory round & slot<>0 after decrementing slot counter	decrement slot counter	arbitrate
	session does not match inventory round	–	arbitrate

Table B.13 *(continued)*

Command	Condition	Action	Next State
QueryAdjust [2]	session matches inventory round & slot=0	backscatter new RN16	reply
	session matches inventory round & slot<>0	–	arbitrate
	session does not match inventory round	–	arbitrate
ACK	all	–	arbitrate
NAK	all	–	arbitrate
Req_RN	all	–	arbitrate
Select	correct parameters	assert or deassert **SL**, or set **inventoried** to *A* or *B*	battery ready
	incorrect parameters	–	arbitrate
Read	all	–	arbitrate
Write	all	–	arbitrate
Kill	all	–	arbitrate
Lock	all	–	arbitrate
Access	all	–	arbitrate
BlockWrite	all	–	arbitrate
BlockErase	all	–	arbitrate
BlockPermalock	all	–	arbitrate
Challenge	supported security timeout, unsupported CSI, not-executable message, nonzero RFU bits	set **C**=0	battery ready
	supported CSI & executable message	store result, set **C**=1	battery ready
Authenticate	all	–	arbitrate
AuthComm	all	–	arbitrate
SecureComm	all	–	arbitrate
ReadBuffer	all	–	arbitrate
KeyUpdate	all	–	arbitrate
Untraceable	all	–	arbitrate
FileSetup	all	–	arbitrate
FileOpen	all	–	arbitrate
FilePrivilege	all	–	arbitrate
TagPrivilege	all	–	arbitrate
FileList	all	–	arbitrate
Flex_Query [1,2]	slot=0; matching **Tag Type Select** criteria, **inventoried** & **SL** flags	backscatter new RN16	reply
	slot<>0; matching **Tag Type Select** criteria, **inventoried** & **SL** flags	–	arbitrate
	otherwise	–	battery ready
BroadcastSync	all	–	arbitrate

Table B.13 *(continued)*

Command	Condition	Action	Next State
HandleSensor	all	–	arbitrate
Faulty	invalid [3]	–	arbitrate
INACT_T [4] or (Selective) Global Timeout	BAP Tag supports Battery Saver Mode	–	stateful sleep or stateful low power listen
	BAP Tag does not support Battery Saver Mode	–	battery ready or stateful battery ready

1: *Query* and *Flex_Query* start a new round and may change the session.

2: *Query, Flex_Query,* and *QueryAdjust* instruct a Tag to load a new random value into its slot counter.

3: "Invalid" shall mean a command not recognizable by the Tag such as (1) an erroneous command (example: a command with an incorrect length field), (2) a command with a CRC error, or (3) an unsupported command.

4: Details about these timers can be found in sub-clauses 7.3.2.2 and 7.3.2.3.

B.2.7 Present state: Reply

Table B.14 — Reply state-transition table

Command	Condition	Action	Next State
Query [1,2]	slot=0; matching **inventoried** & **SL** flags	backscatter new RN16	reply
	slot<>0; matching **inventoried** & **SL** flags	–	arbitrate
	otherwise	–	battery ready
QueryRep	session matches inventory round	–	arbitrate
	session does not match inventory round	–	reply
QueryAdjust [2]	session matches inventory round & slot=0	backscatter new RN16	reply
	session matches inventory round & slot<>0	–	arbitrate
	session does not match inventory round	–	reply
ACK	correct RN16	See Table 6.17	acknowledged
	incorrect RN16	–	arbitrate
NAK	all	–	arbitrate
Req_RN	all	–	arbitrate
Select	correct parameters	assert or deassert **SL**, or set **inventoried** to *A* or *B*	battery ready
	incorrect parameters	–	reply
Read	all	–	arbitrate
Write	all	–	arbitrate
Kill	all	–	arbitrate
Lock	all	–	arbitrate

Table B.14 *(continued)*

Command	Condition	Action	Next State
Access	all	–	arbitrate
BlockWrite	all	–	arbitrate
BlockErase	all	–	arbitrate
BlockPermalock	all	–	arbitrate
Challenge	supported security timeout, unsupported CSI, not-executable message, nonzero RFU bits	set **C**=0	battery ready
	supported CSI & executable message	store result, set **C**=1	battery ready
Authenticate	all	–	arbitrate
AuthComm	all	–	arbitrate
SecureComm	all	–	arbitrate
ReadBuffer	all	–	arbitrate
KeyUpdate	all	–	arbitrate
Untraceable	all	–	arbitrate
FileSetup	all	–	arbitrate
FileOpen	all	–	arbitrate
FilePrivilege	all	–	arbitrate
TagPrivilege	all	–	arbitrate
FileList	all	–	arbitrate
Flex_Query [1,2]	slot=0; matching **Tag Type Select** criteria, **inventoried** & **SL** flags	backscatter new RN16	reply
	slot<>0; matching **Tag Type Select** criteria, **inventoried** & **SL** flags	–	arbitrate
	otherwise	–	battery ready
BroadcastSync	all	–	reply
HandleSensor	all	–	arbitrate
T_2 timeout	See Figure 6.18 and Table 6.16	–	arbitrate
Faulty	invalid [3]	–	reply
INACT_T [4] or (Selective) Global Timeout	BAP Tag supports Battery Saver Mode	–	stateful sleep or stateful low power listen
	BAP Tag does not support Battery Saver Mode	–	battery ready or stateful battery ready

1: *Query* and *Flex_Query* start a new round and may change the session.

2: *Query, Flex_Query,* and *QueryAdjust* instruct a Tag to load a new random value into its slot counter.

3: "Invalid" shall mean a command not recognizable by the Tag such as (1) an erroneous command (example: a command with an incorrect length field), (2) a command with a CRC error, or (3) an unsupported command.

4: Details about these timers can be found in sub-clauses 7.3.2.2 and 7.3.2.3.

B.2.8 Present state: Acknowledged

Table B.15 — Acknowledged state-transition table

Command	Condition	Action	Next State
Query [1]	slot=0; matching **inventoried** [2] & **SL** flags	backscatter new RN16; transition inventoried [2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	reply
	slot<>0; matching **inventoried** [2] & **SL** flags	transition **inventoried** [2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	arbitrate
	otherwise	transition **inventoried** from *A*→*B* or *B*→*A* if and only if new session matches prior session	battery ready
QueryRep	session matches inventory round	transition **inventoried** from *A*→*B* or *B*→*A*	battery ready
	session does not match inventory round	–	acknowledged
QueryAdjust	session matches inventory round	transition **inventoried** from *A*→*B* or *B*→*A*	battery ready
	session does not match inventory round	–	acknowledged
ACK	correct RN16	See Table 6.17	acknowledged
	incorrect RN16	–	arbitrate
NAK	all	–	arbitrate
Req_RN	correct RN16 & access password<>0	backscatter handle	open
	correct RN16 & access password=0	backscatter handle	secured
	incorrect RN16	–	acknowledged
Select	correct parameters	assert or deassert **SL**, or set **inventoried** to *A* or *B*	battery ready
	incorrect parameters	–	acknowledged
Read	all	–	arbitrate
Write	all	–	arbitrate
Kill	all	–	arbitrate
Lock	all	–	arbitrate
Access	all	–	arbitrate
BlockWrite	all	–	arbitrate
BlockErase	all	–	arbitrate
BlockPermalock	all	–	arbitrate
Challenge	supported security timeout, unsupported CSI, not-executable message, nonzero RFU bits	set **C**=0	battery ready
	supported CSI & executable message	store result, set **C**=1	battery ready
Authenticate	all	–	arbitrate
AuthComm	all	–	arbitrate
SecureComm	all	–	arbitrate
ReadBuffer	all	–	arbitrate

Table B.15 *(continued)*

Command	Condition	Action	Next State
KeyUpdate	all	–	arbitrate
Untraceable	all	–	arbitrate
FileSetup	all	–	arbitrate
FileOpen	all	–	arbitrate
FilePrivilege	all	–	arbitrate
TagPrivilege	all	–	arbitrate
FileList	all	–	arbitrate
Flex_Query [1]	slot=0; matching **Tag Type Select** criteria, **inventoried** [2] & **SL** flags	backscatter new RN16; transition inventoried [2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	reply
	slot<>0; matching **Tag Type Select** criteria, **inventoried** [2] & **SL** flags	transition **inventoried** [2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	arbitrate
	otherwise	transition **inventoried** from *A*→*B* or *B*→*A* if and only if new session matches prior session	battery ready
BroadcastSync	all	–	acknowledged
HandleSensor	all	–	arbitrate
T_2 timeout	See Figure 6.18 and Table 6.16	–	arbitrate
Faulty	invalid [3]	–	acknowledged
INACT_T [4] or (Selective) Global Timeout	BAP Tag supports Battery Saver Mode	–	stateful sleep or stateful low power listen
	BAP Tag does not support Battery Saver Mode	–	battery ready or stateful battery ready

1: *Query* and *Flex_Query* start a new round and may change the session. *Query* and *Flex_Query* also instructs a Tag to load a new random value into its slot counter.

2: As described in 6.3.2.10, a Tag transitions its **inventoried** flag prior to evaluating the condition.

3: "Invalid" shall mean a command not recognizable by the Tag such as (1) an erroneous command (example: a command with an incorrect length field), (2) a command with a CRC error, or (3) an unsupported command.

4: Details about these timers can be found in sub-clauses 7.3.2.2 and 7.3.2.3.

B.2.9 Present state: Open

Table B.16 — Open state-transition table

Command	Condition	Action	Next State
Query [1]	slot=0; matching **inventoried** [2] & **SL** flags	backscatter new RN16; transition **inventoried** [2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	reply
	slot<>0; matching **inventoried** [2] & **SL** flags	transition **inventoried** [2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	arbitrate
	otherwise	transition **inventoried** from *A*→*B* or *B*→*A* if and only if new session matches prior session	battery ready
QueryRep	session matches inventory round	transition **inventoried** from *A*→*B* or *B*→*A*	battery ready
	session does not match inventory round	–	open
QueryAdjust	session matches inventory round	transition **inventoried** from *A*→*B* or *B*→*A*	battery ready
	session does not match inventory round	–	open
ACK	correct handle	See Table 6.17	open
	incorrect handle	–	arbitrate
NAK	all	–	arbitrate
Req_RN	all	backscatter new RN16	open
Select	correct parameters	assert or deassert **SL**, or set **inventoried** to *A* or *B*	ready
	incorrect parameters	–	open
Read	all	backscatter data	open
Write	all	backscatter header when done	open
Kill (see also Figure 6.24)	supported security timeout	backscatter error code	open
	password-based kill & correct nonzero kill password	backscatter header when done	killed
	password-based kill & incorrect nonzero kill password	may set security timeout	arbitrate
	password-based kill & kill password=0	backscatter error code	open
	authenticated kill	backscatter error code; may set security timeout	arbitrate
Lock	all	–	open
Access (see also Figure 6.26)	supported security timeout	backscatter error code	open
	correct access password	backscatter handle	secured
	incorrect access password	may set security timeout	arbitrate
BlockWrite	all	backscatter header when done	open
BlockErase	all	backscatter header when done	open
BlockPermalock	all	–	open
Challenge	supported security timeout, unsupported CSI, not-executable message, nonzero RFU bits	set **C**=0	battery ready
	supported CSI and executable message	store result, set **C**=1	battery ready

Table B.16 *(continued)*

Command	Condition	Action	Next State
Authenticate	supported security timeout	backscatter error code	open
	executable & senrep=0	store result; set **C**=1, backscatter response when done	open or secured [3]
	executable & senrep=1	backscatter response when done	open or secured [3]
	crypto error	see crypto suite	arbitrate
	new authentication	reset crypto engine	open
AuthComm	supported security timeout	backscatter error code	open
	prior Tag authentication & executable	backscatter response when done	see encapsulated command
	no prior Tag authentication	backscatter error code	open
	crypto error	see crypto suite	arbitrate
SecureComm	supported security timeout	backscatter error code	open
	prior Tag authentication, executable, & senrep=0	store result; set **C**=1, backscatter response when done	see encapsulated command
	prior Tag authentication, executable, & senrep=1	backscatter response when done	see encapsulated command
	no prior Tag authentication	backscatter error code	open
	crypto error	see crypto suite	arbitrate
ReadBuffer	**C**=1	backscatter data	open
	C=0	backscatter error code	open
KeyUpdate	all	–	open
Untraceable	all	–	open
FileSetup	all	–	open
FileOpen	executable	close current file; open requested file; backscatter file info	open
FilePrivilege	all	–	open
TagPrivilege	all	–	open
FileList	executable & senrep=0	store result; set **C**=1, backscatter response when done	open
	executable & senrep=1	backscatter response when done	open
Flex_Query [1]	slot=0; matching **Tag Type Select** criteria, **inventoried** [2] & **SL** flags	backscatter new RN16; transition **inventoried** [2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	reply
	slot<>0; matching **Tag Type Select** criteria, **inventoried** [2] & **SL** flags	transition **inventoried** [2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	arbitrate
	otherwise	transition **inventoried** from *A*→*B* or *B*→*A* if and only if new session matches prior session	battery ready
BroadcastSync	all	–	open

Table B.16 *(continued)*

Command	Condition	Action	Next State
HandleSensor	valid handle & valid command payload	backscatter response code when internal processing done	open
	valid handle & invalid command payload	backscatter error code	open
	otherwise	–	open
Faulty	unsupported parameters [4]	backscatter error code	open
	incorrect handle [5]	none unless specified by crypto suite	open
	improper [6]	–	arbitrate
	invalid [7]	none unless specified by crypto suite	open
INACT_T [8] or (Selective) Global Timeout	BAP Tag supports Battery Saver Mode	–	stateful sleep or stateful low power listen
	BAP Tag does not support Battery Saver Mode	–	battery ready or stateful battery ready

1: *Query* and *Flex_Query* start a new round and may change the session. *Query* and *Flex_Query* also instructs a Tag to load a new random value into its slot counter.

2: As described in 6.3.2.10, a Tag transitions its **inventoried** flag prior to evaluating the condition.

3: See cryptographic suite

4: "Unsupported parameters" shall mean an access command with a correct handle and CRC and that is recognizable by the Tag but contains or specifies (1) a nonzero or incorrect RFU value, (2) an unsupported CSI; (3) an encapsulated command that is unsupported or disallowed, (4) an unsupported or incorrect memory bank, memory location, address range, or FileNum, (5) a hidden or locked memory bank or location, (6) an unsupported file or files, (7) a command that requires encapsulation but is nonetheless unencapsulated (see Table 6.28), (8) a *delayed* or *in-process* reply and the specified operation causes the Tag to encounter an error, (9) an operation for which the Interrogator has insufficient privileges, (10) an unsupported cryptographic parameter, or (11) other parameters not supported by the Tag.

5: "Incorrect handle" shall mean an access command with a correct CRC and that is recognizable by the Tag but has an incorrect handle. The cryptographic suite indicated by CSI in the prior *Challenge* or *Authenticate* command may specify that a Tag reset its cryptographic engine upon receiving a security command with an incorrect handle.

6: "Improper" shall mean a command (except *Req_RN* or *Query*) that is recognizable by the Tag but is interspersed between successive *Kill* or *Access* commands in a password-based kill or access command sequence respectively (see Figure 6.24 and Figure 6.26).

7: "Invalid" shall mean a command not recognizable by the Tag such as (1) an erroneous command (example: a command with an incorrect length field or a *BlockWrite*/*BlockErase* with a zero-valued WordCount), (2) a command with a CRC error, (3) an unsupported command, or (4) a *Write* command for which the immediately preceding command was not a *Req_RN*. The cryptographic suite indicated by CSI in the prior *Challenge* or *Authenticate* command may specify that a Tag reset its cryptographic engine upon receiving an invalid command.

8: Details about these timers can be found in sub-clauses 7.3.2.2 and 7.3.2.3.

B.2.10 Present state: Secured

Table B.17 — Secured state-transition table

Command	Condition	Action	Next State
Query [1]	slot=0; matching **inventoried** [2] & **SL** flags	backscatter new RN16; transition inventoried [2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	reply
	slot<>0; matching **inventoried** [2] & **SL** flags	transition **inventoried** [2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	arbitrate
	otherwise	transition **inventoried** from *A*→*B* or *B*→*A* if and only if new session matches prior session	battery ready
QueryRep	session matches inventory round	transition **inventoried** from *A*→*B* or *B*→*A*	battery ready
	session does not match inventory round	–	secured
QueryAdjust	session matches inventory round	transition **inventoried** from *A*→*B* or *B*→*A*	battery ready
	session does not match inventory round	–	secured
ACK	correct handle	See Table 6.17	secured
	incorrect handle	–	arbitrate
NAK	all	–	arbitrate
Req_RN	all	backscatter new RN16	secured
Select	correct parameters	assert or deassert **SL**, or set **inventoried** to *A* or *B*	battery ready
	incorrect parameters	–	secured
Read	all	backscatter data	secured
Write	all	backscatter header when done	secured
Kill (see also Figure 6.24)	supported security timeout	backscatter error code	secured
	password-based kill & correct nonzero kill password	backscatter header when done	killed
	password-based kill & incorrect nonzero kill password	may set security timeout	arbitrate
	password-based kill & kill password=0	backscatter error code	secured
	authenticated kill with prior Interrogator authentication & AuthKill privilege	backscatter response when done	killed
	authenticated kill but no prior Interrogator authentication or no AuthKill privilege	backscatter error code; may set security timeout	arbitrate
Lock	all	backscatter header when done	secured
Access (see also Figure 6.26)	supported security timeout	backscatter error code	secured
	correct access password	backscatter handle	secured
	incorrect access password	may set security timeout	arbitrate
BlockWrite	all	backscatter header when done	secured
BlockErase	all	backscatter header when done	secured
BlockPermalock	Read/Lock=0	backscatter permalock bits	secured
	Read/Lock=1	backscatter header when done	secured

Table B.17 *(continued)*

Command	Condition	Action	Next State
Challenge	supported security timeout, unsupported CSI, not-executable message, nonzero RFU bits	set **C**=0	battery ready
	supported CSI and executable message	store result, set **C**=1	battery ready
Authenticate	supported security timeout	backscatter error code	secured
	executable & senrep=0	store result; set **C**=1, backscatter response when done	secured
	executable & senrep=1	backscatter response when done	secured
	crypto error	see crypto suite	arbitrate
	new authentication	reset crypto engine	open
AuthComm	supported security timeout	backscatter error code	secured
	prior Interrogator authentication & executable	backscatter response when done	see encapsulated command
	no prior Interrogator authentication	backscatter error code	secured
	crypto error	see crypto suite	arbitrate
SecureComm	supported security timeout	backscatter error code	secured
	prior Interrogator authentication, executable, & senrep=0	store result; set **C**=1, backscatter response when done	see encapsulated command
	prior Interrogator authentication, executable, & senrep=1	backscatter response when done	see encapsulated command
	no prior Interrogator authentication	backscatter error code	secured
	crypto error	see crypto suite	arbitrate
ReadBuffer	**C**=1	backscatter data	secured
	C=0	backscatter error code	secured
KeyUpdate	supported security timeout	backscatter error code	secured
	prior Interrogator authentication, executable, & senrep=0	store result; set **C**=1, backscatter response when done	secured
	prior Interrogator authentication, executable, & senrep=1	backscatter response when done	secured
	no prior Interrogator authentication	backscatter error code	secured
	crypto error	see crypto suite	arbitrate
Untraceable	executable	backscatter header when done	secured
FileSetup	executable & senrep=0	store result; set **C**=1, backscatter response when done	secured
	executable & senrep=1	backscatter response when done	secured
FileOpen	executable	close current file; open requested file; backscatter file info	secured
FilePrivilege	executable & senrep=0	store result; set **C**=1, backscatter response when done	secured
	executable & senrep=1	backscatter response when done	secured

Table B.17 *(continued)*

Command	Condition	Action	Next State
TagPrivilege	executable & senrep=0	store result; set **C**=1, backscatter response when done	secured
	executable & senrep=1	backscatter response when done	secured
FileList	executable & senrep=0	store result; set **C**=1, backscatter response when done	secured
	executable & senrep=1	backscatter response when done	secured
Flex_Query [1]	slot=0; matching **Tag Type Select** criteria, **inventoried** [2] & **SL** flags	backscatter new RN16; transition inventoried [2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	reply
	slot<>0; matching **Tag Type Select** criteria, **inventoried** [2] & **SL** flags	transition **inventoried** [2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	arbitrate
	otherwise	transition **inventoried** from *A*→*B* or *B*→*A* if and only if new session matches prior session	battery ready
BroadcastSync	all	–	secured
HandleSensor	valid handle & valid command payload	backscatter response code when internal processing done	secured
	valid handle & invalid command payload	backscatter error code	secured
	otherwise	–	secured
Faulty	unsupported parameters [3]	backscatter error code	secured
	incorrect handle [4]	none unless specified by crypto suite	secured or open [4]
	improper [5]	–	arbitrate
	invalid [6]	none unless specified by crypto suite	secured or open [6]
INACT_T [7] or (Selective) Global Timeout	BAP Tag supports Battery Saver Mode	–	stateful sleep or stateful low power listen
	BAP Tag does not support Battery Saver Mode	–	battery ready or stateful battery ready

1: *Query* and *Flex_Query* start a new round and may change the session. *Query* and *Flex_Query* also instructs a Tag to load a new random value into its slot counter.

2: As described in 6.3.2.10, a Tag transitions its inventoried flag prior to evaluating the condition.

3: "Unsupported parameters" shall mean an access command with a correct handle and CRC and that is recognizable by the Tag but contains or specifies (1) a nonzero or incorrect RFU value, (2) an unsupported CSI; (3) an encapsulated command that is unsupported or disallowed, (4) an unsupported or incorrect memory bank, memory location, address range, lock payload, blockpermalock payload, KeyID, or FileNum, (5) a hidden or locked memory bank or location, (6) an unsupported file or files, (7) insufficient or unallocateable memory, (8) an unencrypted message that requires encryption, (9) a command that requires encapsulation but is nonetheless unencapsulated (see Table 6.28), (10) a *delayed* or *in-process* reply and the specified operation causes the Tag to encounter an error, (11) an RFU privilege value, (12) an operation for which the Interrogator has insufficient privileges, (13) an unsupported cryptographic parameter, or (14) other parameters not supported by the Tag.

4: "Incorrect handle" shall mean an access command with a correct CRC and that is recognizable by the Tag but has an incorrect handle. The default next state is **secured**, but the cryptographic suite indicated

by CSI in the prior *Challenge* or *Authenticate* command may specify that a Tag reset its crypto engine and transition to the **open** state upon receiving a security command with an incorrect handle.

5: "Improper" shall mean a command (except *Req_RN* or *Query*) that is recognizable by the Tag but is interspersed between successive *Kill* or *Access* commands in a password-based kill or access command sequence, respectively (see Figure 6.24 and Figure 6.26).

6: "Invalid" shall mean a command not recognizable by the Tag such as (1) an erroneous command (example: a command with an incorrect length field or a *BlockWrite*/*BlockErase* with a zero-valued WordCount), (2) a command with a CRC error, (3) an unsupported command, or (4) a *Write* command for which the immediately preceding command was not a *Req_RN*. The default next state is **secured**, but the cryptographic suite indicated by CSI in the prior *Challenge* or *Authenticate* command may specify that a Tag reset its cryptographic engine and transition to the **open** state upon receiving an invalid command.

7: Details about these timers can be found in sub-clauses 7.3.2.2 and 7.3.2.3.

B.2.11 Present state: Killed

Table B.18 — Killed state-transition table

Command	Condition	Action	Next State
Query	all	–	killed
QueryRep	all	–	killed
QueryAdjust	all	–	killed
ACK	all	–	killed
NAK	all	–	killed
Req_RN	all	–	killed
Select	all	–	killed
Read	all	–	killed
Write	all	–	killed
Kill	all	–	killed
Lock	all	–	killed
Access	all	–	killed
BlockWrite	all	–	killed
BlockErase	all	–	killed
BlockPermalock	all	–	killed
Challenge	all	–	killed
Authenticate	all	–	killed
AuthComm	all	–	killed
SecureComm	all	–	killed
ReadBuffer	all	–	killed
KeyUpdate	all	–	killed
Untraceable	all	–	killed
FileSetup	all	–	killed
FileOpen	all	–	killed
FilePrivilege	all	–	killed
TagPrivilege	all	–	killed
FileList	all	–	killed

Table B.18 *(continued)*

Command	Condition	Action	Next State
Flex_Query	all	–	killed
BroadcastSync	all	–	killed
HandleSensor	all	–	killed
Faulty	all	–	killed
INACT_T [1] or (Selective) Global Timeout	BAP Tag supports Battery Saver Mode	–	killed
	BAP Tag does not support Battery Saver Mode	–	killed

1: Details about these timers can be found in sub-clauses 7.3.2.2 and 7.3.2.3.

B.3 State transition tables for BAP Manchester

B.3.1 Present state: Hibernate

Table B.19 — Hibernate state-transition table

Command	Condition	Action	Next State
Activation command preamble	*all*	–	activation code check

B.3.2 Present state: Activation code check

Table B.20 — Activation code check state-transition table

Command	Condition	Action	Next State
Short Activation command	Invalid Activation Mask	–	stateful hibernate
	Valid Activation Mask or Wildcard (Wildcard authorized)	Clear all timers, set **inventoried** flags to A, deassert **SL**	battery ready
Long Activation command	Invalid Activation Mask	–	stateful hibernate
	Valid Activation Mask or Wildcard (Wildcard authorized), matching flag criteria, Session Locking OFF	Clear all timers, set **inventoried** flags to A, deassert **SL**	battery ready
	Valid Activation Mask or Wildcard (Wildcard authorized), matching flag criteria, Session Locking ON or Interrogator locking in effect	Program activating Session timeout timer	battery ready
	Valid Activation Mask or Wildcard (Wildcard authorized), mismatching flag criteria	–	stateful hibernate
N/A	Flag timer expires	Set that flag to *A* as soon as current operations allow	activation code check

B.3.3 Present state: Stateful Hibernate

Table B.21 — Stateful Hibernate state-transition table

Command/ Event	Condition	Action	Next State
Check flag timers	all timers expired	–	hibernate
	at least one flag timer active	–	stateful hibernate
Activation command preamble	all	–	activation code check
N/A	flag timer expires	Set that flag to *A* as soon as current operations allow	stateful hibernate

B.3.4 Present state: Battery Ready

Table B.22 — Battery Ready state-transition table

Command/Event	Condition	Action	Next State
Activation command preamble	all	Manufacturer option to ignore or "re-activate". If re-activation supported, then set **inventoried** flag to *A* for the active battery ready session, deassert **SL**, and go to **hibernate** mode/ **activation code check** state	activation code check
Query_BAT [a] (Short Activation or Long Activation with Session Locking OFF)	slot=0; matching **inventoried** & **SL** flags and mini-select	backscatter new RN16	reply
	slot<>0; matching **inventoried** & **SL** flags and mini-select	–	arbitrate
	Otherwise	–	battery ready
Query_BAT [a] (Long Activation with Session Locking ON)	Matching Interrogator ID (if Interrogator locking is in effect), slot=0; matching activating session **inventoried** & **SL** flags and mini-select	backscatter new RN16	reply
	Matching Interrogator ID (if Interrogator locking is in effect), slot<>0; matching activating session **inventoried** & **SL** flags and mini-select	–	arbitrate
	Otherwise	–	battery ready
QueryRep	All	–	battery ready
QueryAdjust	All	–	battery ready
ACK	All	–	battery ready
NAK	All	–	battery ready
Req_RN	All	–	battery ready

[a] *Query_BAT* starts a new round and may change the session. *Query_BAT* also instructs a Tag to load a new random value into its slot counter.

[b] "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, or any other command either not recognized or not executable by the Tag.

Table B.22 *(continued)*

Command/Event	Condition	Action	Next State
Select (Short Activation or Long Activation with Session Locking OFF)	All	assert or de-assert **SL**, or set **inventoried** to *A* or *B*	battery ready
Select (Long Activation with Session Locking ON)	Matching Interrogator ID (if Interrogator locking is in effect), matching activating session, or Target is **SL**	assert or de-assert **SL**, or set activating session **inventoried** to *A* or *B*	battery ready
	Otherwise	–	battery ready
Read	All	–	battery ready
Write	All	–	battery ready
Kill	All	–	battery ready
Lock	All	–	battery ready
Access	all	–	battery ready
BlockWrite	all	–	battery ready
BlockErase	All	–	battery ready
BlockPermalock	All	–	battery ready
Challenge	supported security timeout, unsupported CSI, not-executable message, nonzero RFU bits	set **C**=0	battery ready
	supported CSI & executable message	store result, set **C**=1	battery ready
Authenticate	All	–	battery ready
AuthComm	All	–	battery ready
SecureComm	All	–	battery ready
ReadBuffer	All	–	battery ready
KeyUpdate	All	–	battery ready
Untraceable	All	–	battery ready
FileSetup	All	–	battery ready
FileOpen	All	–	battery ready
FilePrivilege	All	–	battery ready
TagPrivilege	All	–	battery ready
FileList	All	–	battery ready
Faulty	invalid [b]	–	battery ready
Broadcast ID	All	–	battery ready
Next	all	–	battery ready

[a] *Query_BAT* starts a new round and may change the session. *Query_BAT* also instructs a Tag to load a new random value into its slot counter.

[b] "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, or any other command either not recognized or not executable by the Tag.

Table B.22 *(continued)*

Command/Event	Condition	Action	Next State
Deactivate_BAT	Matching Interrogator ID (if Interrogator locking is in effect), matching activating session (if session locking is in effect), matching **inventoried** & **SL** flags, Override=0	–	stateful hibernate
	Matching Interrogator ID (if Interrogator locking is in effect), Override=1	Set **inventoried** flags to A, deassert **SL**, and clear timers	stateful hibernate
	Matching Interrogator ID (if Interrogator locking is in effect), matching activating session (if session locking is in effect), mismatching **inventoried** or **SL** flags, Override=0	–	battery ready
	Mismatching Interrogator ID (if Interrogator locking is in effect), mismatching activating session (if session locking is in effect), Override=0	–	battery ready
Multirate_Reset	all	Set **inventoried** flags to A, deassert **SL**, and clear timers	stateful hibernate
Flag timer expires	all	Set that flag to *A* as soon as current operations allow	battery ready
(Selective) Global Timeout or INACT_T	all	Set active session flag to *A* and clear active session flag timeout timer, **SL→~SL**	stateful hibernate
HandleSensor	all	–	battery ready
BroadcastSync	all	–	battery ready
OpRegister Read/ Write	all	–	battery ready
Invalid [b]	all	–	battery ready

[a] *Query_BAT* starts a new round and may change the session. *Query_BAT* also instructs a Tag to load a new random value into its slot counter.

[b] "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, or any other command either not recognized or not executable by the Tag.

B.3.5 Present state: Arbitrate

Table B.23 — Arbitrate state-transition table

Command	Condition	Action	Next State
Activation command preamble	all	Manufacturer option to ignore or "re-activate". If re-activation supported, then set **inventoried** flag to *A* for the active battery ready session, deassert **SL**, and go to **hibernate** mode/ **activation code check** state	activation code check

[a] *Query_BAT* starts a new round and may change the session.

[b] *Query_BAT* and *QueryAdjust* instruct a Tag to load a new random value into its slot counter.

[c] "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than *Query_BAT or Deactivate_BAT*) with a session parameter not matching that of the inventory round currently in progress, or any other command either not recognized or not executable by the Tag.

Table B.23 *(continued)*

Command	Condition	Action	Next State
Query_BAT [a,b] (Short Activation or Long Activation with Session Locking OFF)	slot=0; matching **inventoried** & **SL** flags and mini-select	backscatter new RN16	reply
	slot<>0; matching **inventoried** & **SL** flags and mini-select	–	arbitrate
	Otherwise	–	battery ready
Query_BAT [a,b] (Long Activation with Session Locking ON)	Matching Interrogator ID (if Interrogator locking is in effect), slot=0; matching activating session **inventoried** & **SL** flags and mini-select	backscatter new RN16	reply
	Matching Interrogator ID (if Interrogator locking is in effect), slot<>0; matching activating session **inventoried** & **SL** flags and mini-select	–	arbitrate
	Otherwise	–	arbitrate
QueryRep	Matching Interrogator ID (if Interrogator locking is in effect), slot=0 after decrementing slot counter	backscatter new RN16	reply
	Matching Interrogator ID (if Interrogator locking is in effect), slot<>0 after decrementing slot counter	–	arbitrate
QueryAdjust [b]	Matching Interrogator ID (if Interrogator locking is in effect), slot=0	backscatter new RN16	reply
	Matching Interrogator ID (if Interrogator locking is in effect), slot<>0	–	arbitrate
	Otherwise	–	arbitrate
ACK	all	–	arbitrate
NAK	all	–	arbitrate
Req_RN	all	–	arbitrate
Select (Short Activation or Long Activation with Session Locking OFF)	all	assert or de-assert **SL**, or set **inventoried** to *A* or *B*	battery ready
Select (Long Activation with Session Locking ON)	Matching Interrogator ID (if Interrogator locking is in effect), matching activating session, or Target is **SL**	assert or de-assert **SL**, or set activating session **inventoried** to *A* or *B*	battery ready
	Otherwise	–	arbitrate
Read	all	–	arbitrate
Write	all	–	arbitrate
Kill	all	–	arbitrate
Lock	all	–	arbitrate
Access	all	–	arbitrate
Erase	all	–	arbitrate

[a] *Query_BAT* starts a new round and may change the session.

[b] *Query_BAT* and *QueryAdjust* instruct a Tag to load a new random value into its slot counter.

[c] "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than *Query_BAT or Deactivate_BAT*) with a session parameter not matching that of the inventory round currently in progress, or any other command either not recognized or not executable by the Tag.

Table B.23 *(continued)*

Command	Condition	Action	Next State
BlockWrite	all	–	arbitrate
BlockErase	all	–	arbitrate
BlockPermalock	all	–	arbitrate
Challenge	supported security timeout, unsupported CSI, not-executable message, nonzero RFU bits supported CSI & executable message	set **C**=0, store result, set **C**=1	battery ready
Authenticate	all	–	arbitrate
AuthComm	all	–	arbitrate
SecureComm	all	–	arbitrate
ReadBuffer	all	–	arbitrate
KeyUpdate	all	–	arbitrate
Untraceable	all	–	arbitrate
FileSetup	all	–	arbitrate
FileOpen	all	–	arbitrate
FilePrivilege	all	–	arbitrate
TagPrivilege	all	–	arbitrate
FileList	all	–	arbitrate
Faulty	invalid [c]	–	arbitrate
Broadcast ID	all	–	arbitrate
Next	all	–	arbitrate
Deactivate_BAT	Matching Interrogator ID (if Interrogator locking is in effect), matching activating session (if session locking is in effect), matching **inventoried** & **SL** flags, Override=0	–	stateful hibernate
	Matching Interrogator ID (if Interrogator locking is in effect), Override=1	Set **inventoried** flags to A, deassert **SL**, and clear timers	stateful hibernate
	Matching Interrogator ID (if Interrogator locking is in effect), matching activating session (if session locking is in effect), mismatching **inventoried** or **SL** flags, Override=0	–	battery ready
	otherwise	–	arbitrate
Multirate_Reset	all	Set **inventoried** flags to A, deassert **SL**, and clear timers	stateful hibernate
Flag timer expires	all	Set that flag to *A* as soon as current operations allow	arbitrate
(Selective) Global Timeout or INACT_T	all	Set active session flag to *A* and clear active session flag timeout timer, **SL**→~**SL**	stateful hibernate
HandleSensor	all	–	arbitrate

[a] *Query_BAT* starts a new round and may change the session.

[b] *Query_BAT* and *QueryAdjust* instruct a Tag to load a new random value into its slot counter.

[c] "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than *Query_BAT or Deactivate_BAT*) with a session parameter not matching that of the inventory round currently in progress, or any other command either not recognized or not executable by the Tag.

Table B.23 *(continued)*

Command	Condition	Action	Next State
BroadcastSync	all	–	arbitrate
OpRegister Read/ Write	all	–	arbitrate
Invalid [c]	all	–	arbitrate

[a] *Query_BAT* starts a new round and may change the session.

[b] *Query_BAT* and *QueryAdjust* instruct a Tag to load a new random value into its slot counter.

[c] "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than *Query_BAT or Deactivate_BAT*) with a session parameter not matching that of the inventory round currently in progress, or any other command either not recognized or not executable by the Tag.

B.3.6 Present state: Reply

Table B.24 — Reply state-transition table

Command	Condition	Action	Next State
Activation command preamble	all	Manufacturer option to ignore or "re-activate". If re-activation supported, then set **inventoried** flag to *A* for the active battery ready session, deassert **SL**, and go to **hibernate** mode/**activation code check** state	activation code check
Query_BAT [a,b] (Short Activation or Long Activation with Session Locking OFF)	slot=0; matching **inventoried** & **SL** flags and mini-select	backscatter new RN16	reply
	slot<>0; matching **inventoried** & **SL** flags and mini-select	–	arbitrate
	otherwise	–	battery ready
Query_BAT [a,b] (Long Activation with Session Locking ON)	Matching Interrogator ID (if Interrogator locking is in effect), slot=0; matching activating session **inventoried** & **SL** flags and mini-select	backscatter new RN16	reply
	Matching Interrogator ID (if Interrogator locking is in effect), slot<>0; matching activating session **inventoried** & **SL** flags and mini-select	–	arbitrate
	Otherwise	–	reply
QueryRep	Matching Interrogator ID (if Interrogator locking is in effect)	–	arbitrate
	otherwise	–	reply
QueryAdjust [b]	Matching Interrogator ID (if Interrogator locking is in effect), slot=0	backscatter new RN16	reply
	Matching Interrogator ID (if Interrogator locking is in effect), slot<>0	–	arbitrate
	Otherwise	–	reply

[a] *Query_BAT* starts a new round and may change the session.

[b] *Query_BAT* and *QueryAdjust* instruct a Tag to load a new random value into its slot counter.

[c] "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than *Query_BAT or Deactivate_BAT*) with a session parameter not matching that of the inventory round currently in progress, or any other command either not recognized or not executable by the Tag.

Table B.24 *(continued)*

Command	Condition	Action	Next State
ACK	valid RN16	see Table 6.17, Backscatter SSD if authorized (see 8.5.1)	acknowledged
	invalid RN16	–	reply
NAK	Matching Interrogator ID (if Interrogator locking is in effect)	–	arbitrate
	Otherwise	–	reply
Req_RN	all	–	arbitrate
Select (Short Activation or Long Activation with Session Locking OFF)	all	assert or de-assert **SL**, or set **inventoried** to *A* or *B*	battery ready
Select (Long Activation with Session Locking ON)	Matching Interrogator ID (if Interrogator locking is in effect), matching activating session, or Target is **SL**	assert or de-assert **SL**, or set activating session **inventoried** to *A* or *B*	battery ready
	Otherwise	–	reply
Read	all	–	arbitrate
Write	all	–	arbitrate
Kill	all	–	arbitrate
Lock	all	–	arbitrate
Access	all	–	arbitrate
BlockWrite	all	–	arbitrate
BlockErase	all	–	arbitrate
BlockPermalock	all	–	arbitrate
Challenge	supported security timeout, unsupported CSI, not-executable message, nonzero RFU bits	set **C**=0	battery ready
	supported CSI & executable message	store result, set **C**=1	battery ready
Authenticate	all	–	arbitrate
AuthComm	all	–	arbitrate
SecureComm	all	–	arbitrate
ReadBuffer	all	–	arbitrate
KeyUpdate	all	–	arbitrate
Untraceable	all	–	arbitrate
FileSetup	all	–	arbitrate
FileOpen	all	–	arbitrate
FilePrivilege	all	–	arbitrate
TagPrivilege	all	–	arbitrate
FileList	all	–	arbitrate

[a] *Query_BAT* starts a new round and may change the session.

[b] *Query_BAT* and *QueryAdjust* instruct a Tag to load a new random value into its slot counter.

[c] "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than *Query_BAT or Deactivate_BAT*) with a session parameter not matching that of the inventory round currently in progress, or any other command either not recognized or not executable by the Tag.

Table B.24 *(continued)*

Command	Condition	Action	Next State
T_2 timeout	See Figure 6.18 and Table 6.16	–	arbitrate
Faulty	invalid [c]	–	reply
Broadcast ID	all	–	reply
Next	all	–	reply
Deactivate_BAT	Matching Interrogator ID (if Interrogator locking is in effect), matching activating session (if session locking is in effect), matching **inventoried** & **SL** flags, Override=0	–	stateful hibernate
	Matching Interrogator ID (if Interrogator locking is in effect), Override=1	Set **inventoried** flags to A, deassert **SL**, and clear timers	stateful hibernate
	Matching Interrogator ID (if Interrogator locking is in effect), matching activating session (if session locking is in effect), mismatching **inventoried** or **SL** flags, Override=0	–	battery ready
	otherwise		reply
Multirate_Reset	all	Set **inventoried** flags to A, deassert **SL**, and clear timers	stateful hibernate
Flag timer expires	all	Set that flag to *A* as soon as current operations allow	reply
T_2 timeout	See Figure 6.18 and Table 6.16	–	arbitrate
HandleSensor	all	–	arbitrate
BroadcastSync	all	–	reply
OpRegister Read/ Write	all	–	arbitrate
Invalid [c]	all	–	reply

[a] *Query_BAT* starts a new round and may change the session.

[b] *Query_BAT* and *QueryAdjust* instruct a Tag to load a new random value into its slot counter.

[c] "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than *Query_BAT or Deactivate_BAT*) with a session parameter not matching that of the inventory round currently in progress, or any other command either not recognized or not executable by the Tag.

B.3.7 Present state: Acknowledged

Table B.25 — Acknowledged state-transition table

Command	Condition	Action	Next State
Activation command preamble	all	Manufacturer option to ignore or "re-activate". If re-activation supported, then set **inventoried** flag to *A* for the active battery ready session, deassert **SL**, and go to **hibernate** mode/ **activation code check** state	activation code check
Query_BAT [a] (Short Activation or Long Activation with Session Locking OFF)	slot=0; matching **inventoried** [b] & **SL** flags and mini-select	backscatter new RN16; transition **inventoried** [b] from *A*→*B* or *B*→*A* if and only if new session matches prior session	reply
	slot<>0; matching **inventoried** [b] & **SL** flags and mini-select	transition **inventoried** [b] from *A*→*B* or *B*→*A* if and only if new session matches prior session	arbitrate
	otherwise	transition **inventoried** from *A*→*B* or *B*→*A* if and only if new session matches prior session	battery ready
Query_BAT [a] (Long Activation with Session Locking ON)	Matching Interrogator ID (if Interrogator locking is in effect), slot=0; matching activating session **inventoried** [b] & **SL** flags and mini-select	backscatter new RN16; transition **inventoried** [b] from *A*→*B* or *B*→*A*	reply
	Matching Interrogator ID (if Interrogator locking is in effect), slot<>0; matching activating session **inventoried** [b] & **SL** flags and mini-select	backscatter new RN16; transition **inventoried** [b] from *A*→*B* or *B*→*A*	arbitrate
	otherwise	–	acknowledged
QueryRep	Matching Interrogator ID (if Interrogator locking is in effect)	transition **inventoried** from *A*→*B* or *B*→*A*	battery ready
	otherwise	–	acknowledged
QueryAdjust	Matching Interrogator ID (if Interrogator locking is in effect)	transition **inventoried** from *A*→*B* or *B*→*A*	battery ready
	otherwise	–	acknowledged
ACK	valid RN16	see Table 6.17, Backscatter SSD if authorized (see 8.5.1)	acknowledged
	invalid RN16	–	arbitrate
NAK	Matching Interrogator ID (if Interrogator locking is in effect)	–	arbitrate
	otherwise	–	acknowledged

[a] *Query_BAT* starts a new round and may change the session. *Query_BAT* also instructs a Tag to load a new random value into its slot counter.

[b] As described in 6.3.2.10, a Tag transitions its **inventoried** flag prior to evaluating the condition.

[c] "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than *Query_BAT or Deactivate_BAT*) with a session parameter not matching that of the inventory round currently in progress, or any other command either not recognized or not executable by the Tag.

Table B.25 *(continued)*

Command	Condition	Action	Next State
Req_RN	valid RN16 & access password<>0	backscatter handle	open
	valid RN16 & access password=0	backscatter handle	secured
	invalid RN16	–	acknowledged
Select (Short Activation or Long Activation with Session Locking OFF)	all	assert or de-assert **SL**, or set **inventoried** to *A* or *B*	battery ready
Select (Long Activation with Session Locking ON)	Matching Interrogator ID (if Interrogator locking is in effect), matching activating session, or Target is **SL**	assert or de-assert **SL**, or set activating session **inventoried** to *A* or *B*	battery ready
	Otherwise	–	acknowledged
Read	all	–	arbitrate
Write	all	–	arbitrate
Kill	all	–	arbitrate
Lock	all	–	arbitrate
Access	all	–	arbitrate
BlockWrite	all	–	arbitrate
BlockErase	all	–	arbitrate
BlockPermalock	all	–	arbitrate
Challenge	supported security timeout, unsupported CSI, not-executable message, nonzero RFU bits	set **C**=0	battery ready
	supported CSI & executable message	store result, set **C**=1	battery ready
Authenticate	all	–	arbitrate
AuthComm	all	–	arbitrate
SecureComm	all	–	arbitrate
ReadBuffer	all	–	arbitrate
KeyUpdate	all	–	arbitrate
Untraceable	all	–	arbitrate
FileSetup	all	–	arbitrate
FileOpen	all	–	arbitrate
FilePrivilege	all	–	arbitrate
TagPrivilege	all	–	arbitrate
FileList	all	–	arbitrate
T_2 timeout	See Figure 6.18 and Table 6.16	–	arbitrate
Faulty	invalid [c]	–	acknowledged

[a] *Query_BAT* starts a new round and may change the session. *Query_BAT* also instructs a Tag to load a new random value into its slot counter.

[b] As described in 6.3.2.10, a Tag transitions its **inventoried** flag prior to evaluating the condition.

[c] "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than *Query_BAT or Deactivate_BAT*) with a session parameter not matching that of the inventory round currently in progress, or any other command either not recognized or not executable by the Tag.

Table B.25 *(continued)*

Command	Condition	Action	Next State
Broadcast ID	all	–	acknowledged
Next	valid RN16	Backscatter RN16	stateful hibernate
	invalid RN16	–	acknowledged
Deactivate_BAT	Matching Interrogator ID (if Interrogator locking is in effect), matching activating session (if session locking is in effect), matching **inventoried** & **SL** flags, Override=0	–	stateful hibernate
	Matching Interrogator ID (if Interrogator locking is in effect), Override=1	Set **inventoried** flags to A, deassert **SL**, and clear timers	stateful hibernate
	Matching Interrogator ID (if Interrogator locking is in effect), matching activating session (if session locking is in effect), mismatching **inventoried** or **SL** flags, Override=0	–	battery ready
	otherwise	–	acknowledged
Multirate_Reset	all	Set **inventoried** flags to A, deassert **SL**, and clear timers	stateful hibernate
Flag timer expires	all	Set that flag to *A* as soon as current operations allow	acknowledged
T_2 timeout	See Figure 6.18 and Table 6.16	–	arbitrate
HandleSensor	all	–	arbitrate
BroadcastSync	all	–	acknowledged
OpRegister Read/ Write	all	–	arbitrate
Invalid [c]	all	–	acknowledged

[a] *Query_BAT* starts a new round and may change the session. *Query_BAT* also instructs a Tag to load a new random value into its slot counter.

[b] As described in 6.3.2.10, a Tag transitions its **inventoried** flag prior to evaluating the condition.

[c] "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than *Query_BAT or Deactivate_BAT*) with a session parameter not matching that of the inventory round currently in progress, or any other command either not recognized or not executable by the Tag.

B.3.8 Present state: Open

Table B.26 — Open state-transition table

Command	Condition	Action	Next State
Activation Command Preamble	all	Manufacturer option to ignore or "re-activate". If re-activation supported, then set **inventoried** flag to *A* for the active battery ready session, deassert **SL**, and go to **hibernate** mode/ **activation code check** state	activation code check
Query_BAT [a] (Short Activation or Long Activation with Session Locking OFF)	slot=0; matching **inventoried** [b] & **SL** flags and mini-select	backscatter new RN16; transition **inventoried** [b] from *A*→*B* or *B*→*A* if and only if new session matches prior session	reply
	slot<>0; matching **inventoried** [b] & **SL** flags and mini-select	transition **inventoried** [b] from *A*→*B* or *B*→*A* if and only if new session matches prior session	arbitrate
	otherwise	transition **inventoried** from *A*→*B* or *B*→*A* if and only if new session matches prior session	battery ready
Query_BAT [a] (Long Activation with Session Locking ON)	Matching Interrogator ID (if Interrogator locking is in effect), slot=0; matching activating session **inventoried** [b] & **SL** flags and mini-select	backscatter new RN16; transition **inventoried** [b] from *A*→*B* or *B*→*A*	reply
	Matching Interrogator ID (if Interrogator locking is in effect), slot<>0; matching activating session **inventoried** [b] & **SL** flags and mini-select	backscatter new RN16; transition **inventoried** [b] from *A*→*B* or *B*→*A*	arbitrate
	otherwise	–	open
QueryRep	Matching Interrogator ID (if Interrogator locking is in effect)	transition **inventoried** from *A*→*B* or *B*→*A*	battery ready
	otherwise	–	open
QueryAdjust	Matching Interrogator ID (if Interrogator locking is in effect)	transition **inventoried** from *A*→*B* or *B*→*A*	battery ready
	otherwise	–	open
ACK	valid handle	see Table 6.17, Backscatter SSD if authorized	open
	invalid handle	–	open
NAK	Matching Interrogator ID (if Interrogator locking is in effect)	–	arbitrate
	otherwise	–	open
Req_RN	valid handle	backscatter new RN16	open
	invalid handle	–	open

[a] *Query_BAT* starts a new round and may change the session. *Query_BAT* also instructs a Tag to load a new random value into its slot counter.

[b] As described in 6.3.2.10, a Tag transitions its **inventoried** flag prior to evaluating the condition.

[d] "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than *Query_BAT* or *Deactivate_BAT*) with a session parameter not matching that of the inventory round currently in progress, an otherwise valid command interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively; or any other command either not recognized or not executable by the Tag.

Table B.26 *(continued)*

Command	Condition	Action	Next State
Select (Short Activation or Long Activation with Session Locking OFF)	all	assert or de-assert **SL**, or set **inventoried** to *A* or *B*	battery ready
Select (Long Activation with Session Locking ON)	Matching Interrogator ID (if Interrogator locking is in effect), matching activating session, or Target is **SL**	assert or de-assert **SL**, or set activating session **inventoried** to *A* or *B*	battery ready
	Otherwise	–	open
Read	valid handle & valid memory access	backscatter data and handle	open
	valid handle & invalid memory access	backscatter error code	open
	invalid handle	–	open
Write	valid handle & valid memory access	backscatter handle when done	open
	valid handle & invalid memory access	backscatter error code	open
	invalid handle		open
Kill (see also Figure 6.24)	valid handle & valid nonzero kill password & Recom = 0	backscatter handle when done	killed
	valid handle & valid nonzero kill password & Recom <> 0	backscatter handle when done	open
	valid handle & invalid nonzero kill password	–	arbitrate
	valid handle & kill password=0	backscatter error code	open
	invalid handle	–	open
Lock	all	–	open
Access (see also Figure 6.26)	valid handle & valid access password	backscatter handle	secured
	valid handle & invalid access password	–	arbitrate
	invalid handle	–	open
BlockWrite	valid handle & valid memory access	backscatter handle when done	open
	valid handle & invalid memory access	backscatter error code	open
	invalid handle	–	open
BlockErase	valid handle & valid memory access	backscatter handle when done	open
	valid handle & invalid memory access	backscatter error code	open
	invalid handle	–	open
BlockPermalock	all	–	open
Challenge	supported security timeout, unsupported CSI, not-executable message, nonzero RFU bits	set **C**=0	ready
	supported CSI and executable message	store result, set **C**=1	ready

a *Query_BAT* starts a new round and may change the session. *Query_BAT* also instructs a Tag to load a new random value into its slot counter.

b As described in 6.3.2.10, a Tag transitions its **inventoried** flag prior to evaluating the condition.

d "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than *Query_BAT* or *Deactivate_BAT*) with a session parameter not matching that of the inventory round currently in progress, an otherwise valid command interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively; or any other command either not recognized or not executable by the Tag.

Table B.26 *(continued)*

Command	Condition	Action	Next State
Authenticate	supported security timeout	backscatter error code	open
	executable & senrep=0	store result; set **C**=1, backscatter response when done	open or secured [3]
	executable & senrep=1	backscatter response when done	open or secured [3]
	crypto error	see crypto suite	arbitrate
	new authentication	reset crypto engine	open
AuthComm	supported security timeout	backscatter error code	open
	prior Tag authentication & executable	backscatter response when done	see encapsulated command
	no prior Tag authentication	backscatter error code	open
	crypto error	see crypto suite	arbitrate
SecureComm	supported security timeout	backscatter error code	open
	prior Tag authentication, executable, & senrep=0	store result; set **C**=1, backscatter response when done	see encapsulated command
	prior Tag authentication, executable, & senrep=1	backscatter response when done	see encapsulated command
	no prior Tag authentication	backscatter error code	open
	crypto error	see crypto suite	arbitrate
ReadBuffer	**C**=1	backscatter data	open
	C=0	backscatter error code	open
KeyUpdate	all	–	open
Untraceable	all	–	open
FileSetup	all	–	open
FileOpen	executable	close current file; open requested file; backscatter file info	open
FilePrivilege	all	–	open
TagPrivilege	all	–	open
FileList	executable & senrep=0	store result; set **C**=1, backscatter response when done	open
	executable & senrep=1	backscatter response when done	open

a *Query_BAT* starts a new round and may change the session. *Query_BAT* also instructs a Tag to load a new random value into its slot counter.

b As described in 6.3.2.10, a Tag transitions its **inventoried** flag prior to evaluating the condition.

d "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than *Query_BAT* or *Deactivate_BAT*) with a session parameter not matching that of the inventory round currently in progress, an otherwise valid command interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively; or any other command either not recognized or not executable by the Tag.

Table B.26 *(continued)*

Command	Condition	Action	Next State
Faulty	unsupported parameters [4]	backscatter error code	open
	incorrect handle [c]	none unless specified by crypto suite	open
	improper [6]	–	arbitrate
	invalid [7]	none unless specified by crypto suite	open
Broadcast ID	all	–	open
Next	valid RN16_handle	Backscatter RN16_handle	stateful hibernate
	invalid RN16_handle	–	open
Deactivate_BAT	Matching Interrogator ID (if Interrogator locking is in effect), matching activating session (if session locking is in effect), matching **inventoried** & **SL** flags, Override=0	–	stateful hibernate
	Matching Interrogator ID (if Interrogator locking is in effect), Override=1	Set **inventoried** flags to A, deassert **SL**, and clear timers	stateful hibernate
	Matching Interrogator ID (if Interrogator locking is in effect), matching activating session (if session locking is in effect), mismatching **inventoried** or **SL** flags, Override=0	–	battery ready
	otherwise	–	open
Multirate_Reset	all	Set **inventoried** flags to A, deassert **SL**, and clear timers	stateful hibernate
Flag timer expires	all	Set that flag to *A* as soon as current operations allow	open
(Selective) Global Timeout or INACT_T	all	Set active session flag to *A* and clear active session flag timeout timer, **SL**→~**SL**	stateful hibernate
HandleSensor	valid handle & valid command payload	backscatter header = 0, response code and handle when internal processing done	open
	valid handle & invalid command payload	backscatter header = 1, error code (see Annex I) and handle	open
	invalid handle	–	open
BroadcastSync	all	–	open
OpRegister Read/ Write	all	–	open

[a] *Query_BAT* starts a new round and may change the session. *Query_BAT* also instructs a Tag to load a new random value into its slot counter.

[b] As described in 6.3.2.10, a Tag transitions its **inventoried** flag prior to evaluating the condition.

[d] "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than *Query_BAT* or *Deactivate_BAT*) with a session parameter not matching that of the inventory round currently in progress, an otherwise valid command interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively; or any other command either not recognized or not executable by the Tag.

Table B.26 *(continued)*

Command	Condition	Action	Next State
Invalid [d]	all, excluding valid commands interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively (see Figure 6.24 and Figure 6.26).	–	open
	Otherwise valid commands, except *Req_RN* or *Query_BAT*, interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively (see Figure 6.24 and Figure 6.26).	–	arbitrate

[a] *Query_BAT* starts a new round and may change the session. *Query_BAT* also instructs a Tag to load a new random value into its slot counter.

[b] As described in 6.3.2.10, a Tag transitions its **inventoried** flag prior to evaluating the condition.

[d] "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than *Query_BAT* or *Deactivate_BAT*) with a session parameter not matching that of the inventory round currently in progress, an otherwise valid command interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively; or any other command either not recognized or not executable by the Tag.

B.3.9 Present state: Secured

Table B.27 — Secured state-transition table

Command	Condition	Action	Next State
Activation command preamble	all	Manufacturer option to ignore or "re-activate". If re-activation supported, then set **inventoried** flag to *A* for the active battery ready session, deassert **SL**, and go to **hibernate** mode/ **activation code check** state	activation code check
Query_BAT [a] (Short Activation or Long Activation with Session Locking OFF)	slot=0; matching **inventoried** [b] & **SL** flags and mini-select	backscatter new RN16; transition **inventoried** [b] from *A*→*B* or *B*→*A* if and only if new session matches prior session	reply
	slot<>0; matching **inventoried** [b] & **SL** flags and mini-select	transition **inventoried** [b] from *A*→*B* or *B*→*A* if and only if new session matches prior session	arbitrate
	otherwise	transition **inventoried** from *A*→*B* or *B*→*A* if and only if new session matches prior session	battery ready

[a] *Query_BAT* starts a new round and may change the session. *Query_BAT* also instructs a Tag to load a new random value into its slot counter.

[b] As described in 6.3.2.10, a Tag transitions its **inventoried** flag prior to evaluating the condition.

[d] "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than *Query_BAT* or *Deactivate_BAT*) with a session parameter not matching that of the inventory round currently in progress, an otherwise valid command interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively; or any other command either not recognized or not executable by the Tag.

Table B.27 *(continued)*

Command	Condition	Action	Next State
Query_BAT [a] (Long Activation with Session Locking ON)	Matching Interrogator ID (if Interrogator locking is in effect), slot=0; matching activating session **inventoried** [b] & **SL** flags and mini-select	backscatter new RN16; transition **inventoried** [b] from *A*→*B* or *B*→*A*	reply
	Matching Interrogator ID (if Interrogator locking is in effect), slot<>0; matching activating session **inventoried** [b] & **SL** flags and mini-select	backscatter new RN16; transition **inventoried** [b] from *A*→*B* or *B*→*A*	arbitrate
	otherwise	–	secured
QueryRep	Matching Interrogator ID (if Interrogator locking is in effect)	transition **inventoried** from *A*→*B* or *B*→*A*	battery ready
	otherwise	–	secured
QueryAdjust	Matching Interrogator ID (if Interrogator locking is in effect)	transition **inventoried** from *A*→*B* or *B*→*A*	battery ready
	otherwise		secured
ACK	valid handle	see Table 6.17, Backscatter SSD if authorized (see 8.5.1)	secured
	invalid handle	–	secured
NAK	Matching Interrogator ID (if Interrogator locking is in effect)	–	arbitrate
	otherwise	–	secured
Req_RN	valid handle	backscatter new RN16	secured
	invalid handle	–	secured
Select (Short Activation or Long Activation with Session Locking OFF)	all	assert or de-assert **SL**, or set **inventoried** to *A* or B	battery ready
Select (Long Activation with Session Locking ON)	Matching Interrogator ID (if Interrogator locking is in effect), matching activating session, or Target is **SL**	assert or de-assert **SL**, or set activating session **inventoried** to *A* or *B*	battery ready
	Otherwise	–	secured
Read	valid handle & valid memory access	backscatter data and handle	secured
	valid handle & invalid memory access	backscatter error code	secured
	invalid handle	–	secured
Write	valid handle & valid memory access	backscatter handle when done	secured
	valid handle & invalid memory access	backscatter error code	secured
	invalid handle	–	secured

[a] *Query_BAT* starts a new round and may change the session. *Query_BAT* also instructs a Tag to load a new random value into its slot counter.

[b] As described in 6.3.2.10, a Tag transitions its **inventoried** flag prior to evaluating the condition.

[d] "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than *Query_BAT* or *Deactivate_BAT*) with a session parameter not matching that of the inventory round currently in progress, an otherwise valid command interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively; or any other command either not recognized or not executable by the Tag.

Table B.27 *(continued)*

Command	Condition	Action	Next State
Kill (see also Figure 6.24)	valid handle & valid nonzero kill password & Recom = 0	backscatter handle when done	killed
	valid handle & valid nonzero kill password & Recom <> 0	backscatter handle when done	secured
	valid handle & invalid nonzero kill password	–	arbitrate
	valid handle & kill password=0	backscatter error code	secured
	invalid handle	–	secured
Lock	valid handle & valid lock payload	backscatter handle when done	secured
	valid handle & invalid lock payload	backscatter error code	secured
	invalid handle	–	secured
Access (see also Figure 6.26)	valid handle & valid access password	backscatter handle	secured
	valid handle & invalid access password	–	arbitrate
	invalid handle	–	secured
BlockWrite	valid handle & valid memory access	backscatter handle when done	secured
	valid handle & invalid memory access	backscatter error code	secured
	invalid handle	–	secured
BlockErase	valid handle & valid memory access	backscatter handle when done	secured
	valid handle & invalid memory access	backscatter error code	secured
	invalid handle	–	secured
BlockPermalock	valid handle, valid payload, & Read/Lock = 0	backscatter permalock bits and handle	secured
	valid handle, invalid payload, & Read/Lock = 0	backscatter error code	secured
	valid handle, valid payload, & Read/Lock = 1	backscatter handle when done	secured
	valid handle, invalid payload, & Read/Lock = 1	backscatter error code	secured
	invalid handle	–	secured
Challenge	supported security timeout, unsupported CSI, not-executable message, nonzero RFU bits	set **C**=0	ready
	supported CSI and executable message	store result, set **C**=1	ready

a *Query_BAT* starts a new round and may change the session. *Query_BAT* also instructs a Tag to load a new random value into its slot counter.

b As described in 6.3.2.10, a Tag transitions its **inventoried** flag prior to evaluating the condition.

d "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than *Query_BAT* or *Deactivate_BAT*) with a session parameter not matching that of the inventory round currently in progress, an otherwise valid command interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively; or any other command either not recognized or not executable by the Tag.

Table B.27 *(continued)*

Command	Condition	Action	Next State
Authenticate	supported security timeout	backscatter error code	secured
	executable & senrep=0	store result; set **C**=1, backscatter response when done	secured
	executable & senrep=1	backscatter response when done	secured
	crypto error	see crypto suite	arbitrate
	new authentication	reset crypto engine	open
AuthComm	supported security timeout	backscatter error code	secured
	prior Interrogator authentication & executable	backscatter response when done	see encapsulated command
	no prior Interrogator authentication	backscatter error code	secured
	crypto error	see crypto suite	arbitrate
SecureComm	supported security timeout	backscatter error code	secured
	prior Interrogator authentication, executable, & senrep=0	store result; set **C**=1, backscatter response when done	see encapsulated command
	prior Interrogator authentication, executable, & senrep=1	backscatter response when done	see encapsulated command
	no prior Interrogator authentication	backscatter error code	secured
	crypto error	see crypto suite	arbitrate
ReadBuffer	**C**=1	backscatter data	secured
	C=0	backscatter error code	secured
KeyUpdate	supported security timeout	backscatter error code	secured
	prior Interrogator authentication, executable, & senrep=0	store result; set **C**=1, backscatter response when done	secured
	prior Interrogator authentication, executable, & senrep=1	backscatter response when done	secured
	no prior Interrogator authentication	backscatter error code	secured
	crypto error	see crypto suite	arbitrate
Untraceable	executable	backscatter header when done	secured
FileSetup	executable & senrep=0	store result; set **C**=1, backscatter response when done	secured
	executable & senrep=1	backscatter response when done	secured
FileOpen	executable	close current file; open requested file; backscatter file info	secured

[a] *Query_BAT* starts a new round and may change the session. *Query_BAT* also instructs a Tag to load a new random value into its slot counter.

[b] As described in 6.3.2.10, a Tag transitions its **inventoried** flag prior to evaluating the condition.

[d] "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than *Query_BAT* or *Deactivate_BAT*) with a session parameter not matching that of the inventory round currently in progress, an otherwise valid command interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively; or any other command either not recognized or not executable by the Tag.

Table B.27 *(continued)*

Command	Condition	Action	Next State
FilePrivilege	executable & senrep=0	store result; set **C**=1, backscatter response when done	secured
	executable & senrep=1	backscatter response when done	secured
TagPrivilege	executable & senrep=0	store result; set **C**=1, backscatter response when done	secured
	executable & senrep=1	backscatter response when done	secured
FileList	executable & senrep=0	store result; set **C**=1, backscatter response when done	secured
	executable & senrep=1	backscatter response when done	secured
Faulty	unsupported parameters [3]	backscatter error code	secured
	incorrect handle [4]	none unless specified by crypto suite	secured or open [4]
	improper [5]	–	arbitrate
	invalid [d]	none unless specified by crypto suite	secured or open [6]
Broadcast ID	all	–	secured
Next	valid RN16_handle	Backscatter RN16_handle	stateful hibernate
	invalid RN16_handle	–	Secured
Deactivate_BAT	Matching Interrogator ID (if Interrogator locking is in effect), matching activating session (if session locking is in effect), matching **inventoried** & **SL** flags, Override=0	–	stateful hibernate
	Matching Interrogator ID (if Interrogator locking is in effect), Override=1	Set **inventoried** flags to A, deassert **SL**, and clear timers	stateful hibernate
	Matching Interrogator ID (if Interrogator locking is in effect), matching activating session (if session locking is in effect), mismatching **inventoried** or SL flags, Override=0	–	battery ready
	otherwise	–	secured
Multirate_Reset	all	Set **inventoried** flags to A, deassert **SL**, and clear timers	stateful hibernate
Flag timer expires	all	Set that flag to *A* as soon as current operations allow	secured
(Selective) Global Timeout or INACT_T	all	Set active session flag to *A* and clear active session flag timeout timer, **SL**→~**SL**	stateful hibernate

[a] *Query_BAT* starts a new round and may change the session. *Query_BAT* also instructs a Tag to load a new random value into its slot counter.

[b] As described in 6.3.2.10, a Tag transitions its **inventoried** flag prior to evaluating the condition.

[d] "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than *Query_BAT* or *Deactivate_BAT*) with a session parameter not matching that of the inventory round currently in progress, an otherwise valid command interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively; or any other command either not recognized or not executable by the Tag.

Table B.27 *(continued)*

Command	Condition	Action	Next State
HandleSensor	valid handle & valid command payload	backscatter header = 0, response code and handle when internal processing done	secured
	valid handle & invalid command payload	backscatter header = 1, error code (see Annex I) and handle	secured
	invalid handle	–	secured
BroadcastSync	all	–	secured
OpRegister Read/ Write	valid handle, read, valid reg ID and wordcount	backscatter read bit, handle, and data	secured
	valid handle, write, valid reg ID and wordcount	backscatter write bit, handle when done	secured
	valid handle, invalid reg ID or wordcount	backscatter read/write bit, error code	secured
	invalid handle		secured
Invalid [d]	all, excluding valid commands interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively (see Figure 6.24 and Figure 6.26).	–	secured
	Otherwise valid commands, except *Req_RN* or *Query*, interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively (see Figure 6.24 and Figure 6.26).	–	arbitrate

[a] *Query_BAT* starts a new round and may change the session. *Query_BAT* also instructs a Tag to load a new random value into its slot counter.

[b] As described in 6.3.2.10, a Tag transitions its **inventoried** flag prior to evaluating the condition.

[d] "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than *Query_BAT* or *Deactivate_BAT*) with a session parameter not matching that of the inventory round currently in progress, an otherwise valid command interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively; or any other command either not recognized or not executable by the Tag.

B.3.10 Present state: Killed

Table B.28 — Killed state-transition table

Command	Condition	Action	Next State
Activation Command Preamble	all	–	Killed
Query_BAT	all	–	Killed
QueryRep	all	–	Killed
QueryAdjust	all	–	Killed
ACK	all	–	Killed
NAK	all	–	Killed
Req_RN	all	–	killed
Select	all	–	killed

[a] "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, or any other command either not recognized or not executable by the Tag.

Table B.28 *(continued)*

Command	Condition	Action	Next State
Read	all	–	killed
Write	all	–	killed
Kill	all	–	killed
Lock	all	–	killed
Access	all	–	killed
BlockWrite	all	–	killed
BlockErase	all	–	killed
BlockPermalock	all	–	killed
Challenge	all	–	killed
Authenticate	all	–	killed
AuthComm	all	–	killed
SecureComm	all	–	killed
ReadBuffer	all	–	killed
KeyUpdate	all	–	killed
Untraceable	all	–	killed
FileSetup	all	–	killed
FileOpen	all	–	killed
FilePrivilege	all	–	killed
TagPrivilege	all	–	killed
FileList	all	–	killed
Faulty	all	–	killed
Broadcast ID	all	–	killed
Next	all	–	killed
Deactivate_BAT	all	–	killed
Flag timer expires	all	Set that flag to *A* as soon as current operations allow	killed
(Selective) Global Timeout or HACT_T	all	–	killed
Multirate_Reset	all	–	killed
HandleSensor	all	–	killed
BroadcastSync	all	–	killed
OpRegister Read/ Write	all	–	killed
Invalid [a]	all	–	killed

[a] "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, or any other command either not recognized or not executable by the Tag.

Annex C
(normative)

Command-Response Tables

Command-response tables C.1 to C.30 shall define a Tag's response to Interrogator commands for passive PIE tags. For BAP PIE: see C.2 and for BAP Manchester: see C.3

The term "handle" used in the command-response tables is defined in 6.3.2.6.5; error codes are defined in Table I.2; "slot" is the slot-counter output shown in Figure 6.21 and detailed in Annex J; "–" in the "Response" column means that a Tag neither executes the command nor backscatters a reply.

C.1 Command response tables for passive

C.1.1 Command response: Power-up

Table C.1 — Power-up command-response table

Starting State	Condition	Response	Next State
ready, arbitrate, reply, acknowledged, open, secured	power-up	–	ready
killed	all	–	killed

C.1.2 Command response: *Query*

Table C.2 — *Query* [1] command-response table

Starting State	Condition	Response	Next State
ready, arbitrate, reply	slot=0; matching **inventoried** & **SL** flags	backscatter new RN16	reply
	slot<>0; matching **inventoried** & **SL** flags	–	arbitrate
	otherwise	–	ready
acknowledged, open, secured	slot=0; matching **inventoried** [2] & **SL** flags	backscatter new RN16; transition **inventoried** [2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	reply
	slot<>0; matching **inventoried** [2] & **SL** flags	transition **inventoried** [2] from *A*→*B* or *B*→*A* if and only if new session matches prior session	arbitrate
	otherwise	transition **inventoried** from *A*→*B* or *B*→*A* if and only if new session matches prior session	ready
killed	all	–	killed

1: *Query* (in any state other than **killed**) starts a new round and may change the session; *Query* also instructs a Tag to load a new random value into its slot counter.

2: As described in 6.3.2.8, **a Tag transitions its inventoried** flag prior to evaluating the condition.

C.1.3 Command response: *QueryRep*

Table C.3 — *QueryRep* command-response table

Starting State	Condition	Response	Next State
ready	all	–	ready
arbitrate	session matches inventory round & slot=0 after decrementing slot counter	decrement slot counter; backscatter new RN16	reply
	session matches inventory round & slot<>0 after decrementing slot counter	decrement slot counter	arbitrate
	session does not match inventory round	–	arbitrate
reply	session matches inventory round	–	arbitrate
	session does not match inventory round	–	reply
acknowledged	session matches inventory round	transition **inventoried** from *A*→*B* or *B*→*A*	ready
	session does not match inventory round	–	acknowledged
open	session matches inventory round	transition **inventoried** from *A*→*B* or *B*→*A*	ready
	session does not match inventory round	–	open
secured	session matches inventory round	transition **inventoried** from *A*→*B* or *B*→*A*	ready
	session does not match inventory round	–	secured
killed	all	–	killed

C.1.4 Command response: *QueryAdjust*

Table C.4 — *QueryAdjust*[1] command-response table

Starting State	Condition	Response	Next State
ready	all	–	ready
arbitrate	session matches inventory round & slot=0	backscatter new RN16	reply
	session matches inventory round & slot<>0	–	arbitrate
	session does not match inventory round	–	arbitrate
reply	session matches inventory round & slot=0	backscatter new RN16	reply
	session matches inventory round & slot<>0	–	arbitrate
	session does not match inventory round	–	reply
acknowledged	session matches inventory round	transition **inventoried** from *A*→*B* or *B*→*A*	ready
	session does not match inventory round	–	acknowledged

Table C.4 *(continued)*

Starting State	Condition	Response	Next State
open	session matches inventory round	transition **inventoried** from *A*→*B* or *B*→*A*	ready
	session does not match inventory round	–	open
secured	session matches inventory round	transition **inventoried** from *A*→*B* or *B*→*A*	ready
	session does not match inventory round	–	secured
killed	all	–	ready

1: *QueryAdjust*, in the **arbitrate** or **reply** states, instructs a Tag to load a new random value into its slot counter.

C.1.5 Command response: *ACK*

Table C.5 — *ACK* command-response table

Starting State	Condition	Response	Next State
ready	all	–	ready
arbitrate	all	–	arbitrate
reply, acknowledged	correct RN16	see Table 6.17	acknowledged
	incorrect RN16	–	arbitrate
open	correct handle	see Table 6.17	open
	incorrect handle	–	arbitrate
secured	correct handle	see Table 6.17	secured
	incorrect handle	–	arbitrate
killed	all	–	killed

C.1.6 Command response: *NAK*

Table C.6 — *NAK* command-response table

Starting State	Condition	Response	Next State
ready	all	–	ready
arbitrate, reply, acknowledged, open, secured	all	–	arbitrate
killed	all	–	killed

C.1.7 Command response: *Req_RN*

Table C.7 — *Req_RN* command-response table

Starting State	Condition	Response	Next State
ready	all	–	ready
arbitrate, reply	all	–	arbitrate
acknowledged	correct RN16 & access password<>0	backscatter handle	open
	correct RN16 & access password=0	backscatter handle	secured
	incorrect RN16	–	acknowledged
open [1]	all	backscatter new RN16	open
secured [1]	all	backscatter new RN16	secured
killed	all	–	killed

1: See Table C.30 for the Tag response to an incorrect handle.

C.1.8 Command response: *Select*

Table C.8 — *Select* command-response table

Starting State	Condition	Response	Next State
ready	correct parameters	assert or deassert **SL**, or set **inventoried** to *A* or *B*	ready
	incorrect parameters	–	ready
arbitrate	correct parameters	assert or deassert **SL**, or set **inventoried** to *A* or *B*	ready
	incorrect parameters	–	arbitrate
reply	correct parameters	assert or deassert **SL**, or set **inventoried** to *A* or *B*	ready
	incorrect parameters	–	reply
acknowledged	correct parameters	assert or deassert **SL**, or set **inventoried** to *A* or *B*	ready
	incorrect parameters	–	acknowledged
open	correct parameters	assert or deassert **SL**, or set **inventoried** to *A* or *B*	ready
	incorrect parameters	–	open
secured	correct parameters	assert or deassert **SL**, or set **inventoried** to *A* or *B*	ready
	incorrect parameters	–	secured
killed	all	–	killed

C.1.9 Command response: *Read*

Table C.9 — *Read* command-response table

Starting State	Condition	Response	Next State
ready	all	–	ready
arbitrate, reply, acknowledged	all	–	arbitrate
open [1]	all	backscatter data	open
secured [1]	all	backscatter data	secured
killed	all	–	killed

1: See Table C.30 for the Tag response to an incorrect handle or unsupported parameters.

C.1.10 Command response: *Write*

Table C.10 — *Write* command-response table

Starting State	Condition	Response	Next State
ready	all	–	ready
arbitrate, reply, acknowledged	all	–	arbitrate
open [1]	all	backscatter header when done	open
secured [1]	all	backscatter header when done	secured
killed	all	–	killed

1: See Table C.30 for the Tag response to an incorrect handle, unsupported parameters, or an improper command.

C.1.11 Command response: *Kill*

Table C.11 — *Kill* [1] command-response table

Starting State	Condition	Response	Next State
ready	all	–	ready
arbitrate, reply, acknowledged	all	–	arbitrate
open [2]	supported security timeout	backscatter error code	open
	password-based kill & correct nonzero kill password	backscatter header when done	killed
	password-based kill & incorrect nonzero kill password	may set security timeout	arbitrate
	password-based kill & kill password=0	backscatter error code	open
	authenticated kill	backscatter error code; may set security timeout	arbitrate

Table C.11 *(continued)*

Starting State	Condition	Response	Next State
secured [2]	supported security timeout	backscatter error code	secured
	password-based kill & correct nonzero kill password	backscatter header when done	killed
	password-based kill & incorrect nonzero kill password	may set security timeout	arbitrate
	password-based kill & kill password=0	backscatter error code	secured
	authenticated kill with prior Interrogator authentication & AuthKill privilege	backscatter response when done	killed
	authenticated kill but no prior Interrogator authentication or no AuthKill privilege	backscatter error code; may set security timeout	arbitrate
killed	all	–	killed

1: See also Figure 6.24.

2: See Table C.30 for the Tag response to an incorrect handle, unsupported parameters, or an improper command.

C.1.12 Command response: *Lock*

Table C.12 — *Lock* command-response table

Starting State	Condition	Response	Next State
ready	all	–	ready
arbitrate, reply, acknowledged	all	–	arbitrate
open	all	–	open
secured [1]	all	backscatter header when done	secured
killed	all	–	killed

1: See Table C.30 for the Tag response to an incorrect handle or unsupported parameters.

C.1.13 Command response: *Access*

Table C.13 — *Access* [1] command-response table

Starting State	Condition	Response	Next State
ready	all	–	ready
arbitrate, reply, acknowledged	all	–	arbitrate
open [2]	supported security timeout	backscatter error code	open
	correct access password	backscatter handle	secured
	incorrect access password	may set security timeout	arbitrate
secured [2]	supported security timeout	backscatter error code	secured
	correct access password	backscatter handle	secured
	incorrect access password	may set security timeout	arbitrate
killed	all	–	killed

1: See also Figure 6.26.

2: See Table C.30 for the Tag response to an incorrect handle or an improper command.

C.1.14 Command response: *BlockWrite*

Table C.14 — *BlockWrite* command-response table

Starting State	Condition	Response	Next State
ready	all	–	ready
arbitrate, reply, acknowledged	all	–	arbitrate
open [1]	all	backscatter header when done	open
secured [1]	all	backscatter header when done	secured
killed	all	–	killed

1: See Table C.30 for the Tag response to an incorrect handle or unsupported parameters.

C.1.15 Command response: *BlockErase*

Table C.15 — *BlockErase* command-response table

Starting State	Condition	Response	Next State
ready	all	–	ready
arbitrate, reply, acknowledged	all	–	arbitrate
open [1]	all	backscatter header when done	open
secured [1]	all	backscatter header when done	secured
killed	all	–	killed

1: See Table C.30 for the Tag response to an incorrect handle or unsupported parameters.

C.1.16 Command response: *BlockPermalock*

Table C.16 — *BlockPermalock* command-response table

Starting State	Condition	Response	Next State
ready	all	–	ready
arbitrate, reply, acknowledged	all	–	arbitrate
open	all	–	open
secured [1]	Read/Lock=0	backscatter permalock bits	secured
	Read/Lock=1	backscatter header when done	secured
killed	all	–	killed

1: See Table C.30 for the Tag response to an incorrect handle or unsupported parameters.

C.1.17 Command response: *Challenge*

Table C.17 — *Challenge* command-response table

Starting State	Condition	Response	Next State
ready, arbitrate, reply, acknowledged, open, secured	supported security timeout, unsupported CSI, not-executable message, nonzero RFU bits	set **C**=0	ready
	supported CSI and executable message	store result, set **C**=1	ready
killed	all	–	killed

C.1.18 Command response: *Authenticate*

Table C.18 — *Authenticate* command-response table

Starting State	Condition	Response	Next State
ready	all	–	ready
arbitrate, reply, acknowledged	all	–	arbitrate
open [1]	supported security timeout	backscatter error code	open
	executable & senrep=0	store result; set **C**=1, backscatter response when done	open or secured [2]
	executable & senrep=1	backscatter response when done	open or secured [2]
	crypto error	see crypto suite	arbitrate
	new authentication	reset crypto engine	open
secured [1]	supported security timeout	backscatter error code	secured
	executable & senrep=0	store result; set **C**=1, backscatter response when done	secured
	executable & senrep=1	backscatter response when done	secured
	crypto error	see crypto suite	arbitrate
	new authentication	reset crypto engine	open
killed	all	–	killed

1: See Table C.30 for the Tag response to an incorrect handle or unsupported parameters.

2: See cryptographic suite.

C.1.19 Command response: *AuthComm*

Table C.19 — *AuthComm* command-response table

Starting State	Condition	Response	Next State
ready	all	–	ready
arbitrate, reply, acknowledged	all	–	arbitrate

Table C.19 *(continued)*

Starting State	Condition	Response	Next State
open [1]	supported security timeout	backscatter error code	open
	prior Tag authentication & executable	backscatter response when done	see encapsulated command
	no prior Tag authentication	backscatter error code	open
	crypto error	see crypto suite	arbitrate
secured [1]	supported security timeout	backscatter error code	secured
	prior Interrogator authentication & executable	backscatter response when done	see encapsulated command
	no prior Interrogator authentication	backscatter error code	secured
	crypto error	see crypto suite	arbitrate
killed	all	–	killed

1: See Table C.30 for the Tag response to an incorrect handle or unsupported parameters.

C.1.20 Command response: *SecureComm*

Table C.20 — *SecureComm* command-response table

Starting State	Condition	Response	Next State
ready	all	–	ready
arbitrate, reply, acknowledged	all	–	arbitrate
open [1]	supported security timeout	backscatter error code	open
	prior Tag authentication, executable, & senrep=0	store result; set **C**=1, backscatter response when done	see encapsulated command
	prior Tag authentication, executable, & senrep=1	backscatter response when done	see encapsulated command
	no prior Tag authentication	backscatter error code	open
	crypto error	see crypto suite	arbitrate
secured [1]	supported security timeout	backscatter error code	secured
	prior Interrogator authentication, executable, & senrep=0	store result; set **C**=1, backscatter response when done	see encapsulated command
	prior Interrogator authentication, executable, & senrep=1	backscatter response when done	see encapsulated command
	no prior Interrogator authentication	backscatter error code	secured
	crypto error	see crypto suite	arbitrate
killed	all	–	killed

1: See Table C.30 for the Tag response to an incorrect handle or unsupported parameters.

C.1.21 Command response: *ReadBuffer*

Table C.21 — *ReadBuffer* command-response table

Starting State	Condition	Response	Next State
ready	all	–	ready
arbitrate, reply, acknowledged	all	–	arbitrate
open [1]	**C**=1	backscatter data	open
	C=0	backscatter error code	open
secured [1]	**C**=1	backscatter data	secured
	C=0	backscatter error code	secured
killed	all	–	killed

1: See Table C.30 for the Tag response to an incorrect handle or unsupported parameters.

C.1.22 Command response: *KeyUpdate*

Table C.22 — *KeyUpdate* command-response table

Starting State	Condition	Response	Next State
ready	all	–	ready
arbitrate, reply, acknowledged	all	–	arbitrate
open	all	–	open
secured [1]	supported security timeout	backscatter error code	secured
	prior Interrogator authentication, executable, & senrep=0	store result; set **C**=1, backscatter response when done	secured
	prior Interrogator authentication, executable, & senrep=1	backscatter response when done	secured
	no prior Interrogator authentication	backscatter error code	secured
	crypto error	see crypto suite	arbitrate
killed	all	–	killed

1: See Table C.30 for the Tag response to an incorrect handle or unsupported parameters.

C.1.23 Command response: *Untraceable*

Table C.23 — *Untraceable* command-response table

Starting State	Condition	Response	Next State
ready	all	–	ready
arbitrate, reply, acknowledged	all	–	arbitrate
open	all	–	open
secured [1]	executable	backscatter header when done	secured
killed	all	–	killed

1: See Table C.30 for the Tag response to an incorrect handle or unsupported parameters.

C.1.24 Command response: *FileSetup*

Table C.24 — *FileSetup* command-response table

Starting State	Condition	Response	Next State
ready	all	–	ready
arbitrate, reply, acknowledged	all	–	arbitrate
open	all	–	open
secured [1]	executable & senrep=0	store result; set **C**=1, backscatter response when done	secured
	executable & senrep=1	backscatter response when done	secured
killed	all	–	killed

1: See Table C.30 for the Tag response to an incorrect handle or unsupported parameters.

C.1.25 Command response: *FileOpen*

Table C.25 — *FileOpen* command-response table

Starting State	Condition	Response	Next State
ready	all	–	ready
arbitrate, reply, acknowledged	all	–	arbitrate
open [1]	executable	close current file; open requested file; backscatter file info	open
secured [1]	executable	close current file; open requested file; backscatter file info	secured
killed	all	–	killed

1: See Table C.30 for the Tag response to an incorrect handle or unsupported parameters.

C.1.26 Command response: *FilePrivilege*

Table C.26 — *FilePrivilege* command-response table

Starting State	Condition	Response	Next State
ready	all	–	ready
arbitrate, reply, acknowledged	all	–	arbitrate
open	all	–	open
secured [1]	executable & senrep=0	store result; set **C**=1, backscatter response when done	secured
	executable & senrep=1	backscatter response when done	secured
killed	all	–	killed

1: See Table C.30 for the Tag response to an incorrect handle or unsupported parameters.

C.1.27 Command response: *TagPrivilege*

Table C.27 — *TagPrivilege* command-response table

Starting State	Condition	Response	Next State
ready	all	–	ready
arbitrate, reply, acknowledged	all	–	arbitrate
open	all	–	open
secured [1]	executable & senrep=0	store result; set **C**=1, backscatter response when done	secured
	executable & senrep=1	backscatter response when done	secured
killed	all	–	killed

1: See Table C.30 for the Tag response to an incorrect handle or unsupported parameters.

C.1.28 Command response: *FileList*

Table C.28 — *FileList* command-response table

Starting State	Condition	Response	Next State
ready	all	–	ready
arbitrate, reply, acknowledged	all	–	arbitrate
open [1]	executable & senrep=0	store result; set **C**=1, backscatter response when done	open
	executable & senrep=1	backscatter response when done	open
secured [1]	executable & senrep=0	store result; set **C**=1, backscatter response when done	secured
	executable & senrep=1	backscatter response when done	secured
killed	all	–	killed

1: See Table C.30 for the Tag response to an incorrect handle or unsupported parameters.

C.1.29 Command response: T_2 timeout

Table C.29 — T_2 timeout command-response table

Starting State	Condition	Response	Next State
ready	all	–	ready
arbitrate	all	–	arbitrate
reply, acknowledged	See Figure 6.18 and Table 6.16	–	arbitrate
open	all	–	open
secured	all	–	secured
killed	all	–	killed

C.1.30 Command response: Faulty command

Table C.30 — Faulty command-response table

Starting State	Condition	Response	Next State
ready	invalid [1]	–	ready
arbitrate	invalid [1]	–	arbitrate
reply	invalid [1]	–	reply
acknowledged	invalid [1]	–	acknowledged
open	unsupported parameters [2]	backscatter error code	open
	incorrect handle [4]	none unless specified by crypto suite	open
	improper [6]	–	arbitrate
	invalid [7]	none unless specified by crypto suite	open
secured	unsupported parameters [3]	backscatter error code	secured
	incorrect handle [5]	none unless specified by crypto suite	secured or open [9]
	improper [6]	–	arbitrate
	invalid [8]	none unless specified by crypto suite	secured or open [9]
killed	all	–	killed

1: "Invalid" shall mean a command not recognizable by the Tag such as (1) an erroneous command (example: a command with an incorrect length field), (2) a command with a CRC error, or (3) an unsupported command.

2: "Unsupported parameters" shall mean an access command with a correct handle and CRC and that is recognizable by the Tag but contains or specifies (1) a nonzero or incorrect RFU value, (2) an unsupported CSI; (3) an encapsulated command that is unsupported or disallowed, (4) an unsupported or incorrect memory bank, memory location, address range, or FileNum, (5) a hidden or locked memory bank or location, (6) an unsupported file or files, (7) a command that requires encapsulation but is nonetheless unencapsulated (see Table 6.28), (8) a *delayed* or *in-process* reply and the specified operation causes the Tag to encounter an error, (9) an operation for which the Interrogator has insufficient privileges, (10) an unsupported cryptographic parameter, or (11) other parameters not supported by the Tag.

3: "Unsupported parameters" shall mean an access command with a correct handle and CRC and that is recognizable by the Tag but contains or specifies (1) a nonzero or incorrect RFU value, (2) an unsupported CSI; (3) an encapsulated command that is unsupported or disallowed, (4) an unsupported or incorrect memory bank, memory location, address range, lock payload, blockpermalock payload, KeyID, or FileNum, (5) a hidden or locked memory bank or location, (6) an unsupported file or files, (7) insufficient or unallocateable memory, (8) an unencrypted message that requires encryption, (9) a command that requires encapsulation but is nonetheless unencapsulated (see Table 6.28), (10) a *delayed* or *in-process* reply and the specified operation causes the Tag to encounter an error, (11) an RFU privilege value, (12) an operation for which the Interrogator has insufficient privileges, (13) an unsupported cryptographic parameter, or (14) other parameters not supported by the Tag.

4: "Incorrect handle" shall mean an access command with a correct CRC and that is recognizable by the Tag but has an incorrect handle. The cryptographic suite indicated by CSI in the prior *Challenge* or *Authenticate* command may specify that a Tag reset its cryptographic engine upon receiving a security command with an incorrect handle.

5: "Incorrect handle" shall mean an access command with a correct CRC and that is recognizable by the Tag but has an incorrect handle. The default next state is **secured**, but the cryptographic suite indicated by CSI in the prior *Challenge* or *Authenticate* command may specify that a Tag reset its crypto engine and transition to the **open** state upon receiving a security command with an incorrect handle.

6: "Improper" shall mean a command (except *Req_RN* or *Query*) that is recognizable by the Tag but is interspersed between successive *Kill* or *Access* commands in a password-based kill or access command sequence, respectively (see Figure 6.24 and Figure 6.26).

7: "Invalid" shall mean a command not recognizable by the Tag such as (1) an erroneous command (example: a command with an incorrect length field or a *BlockWrite*/*BlockErase* with a zero-valued WordCount), (2) a command with a CRC error, (3) an unsupported command, or (4) a *Write* command for which the immediately preceding command was not a *Req_RN*. The cryptographic suite indicated by CSI in the prior *Challenge* or *Authenticate* command may specify that a Tag reset its cryptographic engine upon receiving an invalid command.

8: "Invalid" shall mean a command not recognizable by the Tag such as (1) an erroneous command (example: a command with an incorrect length field or a *BlockWrite*/*BlockErase* with a zero-valued WordCount), (2) a command with a CRC error, (3) an unsupported command, or (4) a *Write* command for which the immediately preceding command was not a *Req_RN*.. The default next state is **secured**, but the cryptographic suite indicated by CSI in the prior *Challenge* or *Authenticate* command may specify that a Tag reset its cryptographic engine and transition to the **open** state upon receiving an invalid command.

9: See cryptographic suite.

C.2 Command response tables for BAP PIE

Same as in C.1, with the **ready** state replaced by **battery ready**, and the addition of *Flex_Query*, *BroadcastSync*, INACT_T, and (Selective) Global Timeout.

C.2.1 Command response: Flex_Query

Table C.31 — *Flex_Query*[a,c] command-response table

Starting State	Condition	Response	Next State
battery ready, arbitrate, reply	slot=0; matching **inventoried** & **SL** flags	backscatter new RN16	**reply**
	slot<>0; matching **inventoried** & **SL** flags	–	**arbitrate**
	otherwise	–	**battery ready**
acknowledged, open, secured	slot=0; matching **inventoried** [b] & **SL** flags	backscatter new RN16; transition **inventoried** [b] from *A*→*B* or *B*→*A* if and only if new session matches prior session	**reply**
	slot<>0; matching **inventoried** [b] & **SL** flags	transition **inventoried** [b] from *A*→*B* or *B*→*A* if and only if new session matches prior session	**arbitrate**
	all other conditions	transition **inventoried** from *A*→*B* or *B*→*A* if and only if new session matches prior session	**ready**
	otherwise	–	**battery ready**
killed	all	–	**killed**

[a] *Flex_Query* (in any state other than killed) starts a new round and may change the session; *Flex_Query* also instructs a Tag to load a new random value into its slot counter.

[b] As described in 6.3.2.10, a Tag transitions its **inventoried** flag prior to evaluating the condition.

[c] Tag checks the Tag Type Select field, if it matches the conditions in that field, Tag executes this command, otherwise, it ignores it.

C.2.2 Command response: INACT_T or Selective Global Timeout

Table C.32 — INACT_T [a] or Selective Global Timeout [b] command-response table

Starting State	Condition	Response	Next State
battery ready, arbitrate, open, secured	BAP Tag supports Battery Saver Mode	–	**stateful sleep** or **stateful low power listen**
battery ready, arbitrate, open, secured	BAP Tag does not support Battery Saver Mode	–	**battery ready** or **stateful battery ready**
killed	all	–	**killed**

[a] See clause 7.3.2.2 for details on the refresh of the INACT_T timer

[b] See clause 7.3.2.3 for details on the refresh of the Selective Global Timeout timer

C.2.3 Command response: Global Timeout

Table C.33 — Global Timeout command-response table

Starting State	Condition	Response	Next State
battery ready, arbitrate, reply, acknowledged, open, secured	BAP Tag supports Battery Saver Mode	–	**stateful sleep** or **stateful low power listen**
battery ready, arbitrate, reply, acknowledged, open, secured	BAP Tag does not support Battery Saver Mode	–	**battery ready** or **stateful battery ready**
killed	all	–	**killed**

C.2.4 Command response: *HandleSensor*

Table C.34 — *HandleSensor* command-response table

Starting State	Condition	Response	Next State
battery ready	All	–	**battery ready**
arbitrate, reply, acknowledged	All	–	**arbitrate**
open, secured	valid handle & valid command payload	backscatter header = 0, response code [a] and handle when internal processing done	**open, secured**
	valid handle & invalid command payload	backscatter header = 1, error code (see Annex I) and handle	**open, secured**
	invalid handle	–	**open, secured**
killed	All	–	**killed**

[a] Sensor specific output of *HandleSensor* command payload processing.

C.2.5 Command response: *BroadcastSync*

Table C.35 — *BroadcastSync* command-response table

Starting State	Condition	Response	Next State
hibernate,	all	–	**hibernate**
stateful hibernate	all	–	**stateful hibernate**
battery ready, arbitrate, reply, acknowledged, open, secured	all	–	**battery ready, arbitrate, reply, acknowledged, open, secured**
killed	All	–	**killed**

C.3 Command Response Tables for Manchester

C.3.1 Command response: Power-up

Table C.36 — Power-up command-response table

Starting State	Condition	Response	Next State
OFF	power-up	–	**hibernate**
killed	all	–	**killed**

C.3.2 Command response: *QueryRep*

Table C.37 — *QueryRep* command-response table [a]

Starting State	Condition	Response	Next State
battery ready	All	–	**battery ready**
arbitrate	slot<>0 after decrementing slot counter	–	**arbitrate**
	slot=0 after decrementing slot counter	backscatter new RN16	**reply**
reply	All	–	**arbitrate**
acknowledged, open, secured	All	transition **inventoried** from *A*→*B* or *B*→*A*	**battery ready**
killed	All	–	**killed**

[a] In Manchester with Long Activation and Interrogator locking in effect, the Tag checks first the Interrogator ID, if it matches, it proceeds with the command execution, otherwise, it ignores the command.

C.3.3 Command response: *QueryAdjust*

Table C.38 — *QueryAdjust* [a] command-response table [b]

Starting State	Condition	Response	Next State
battery ready	all	–	**battery ready**
arbitrate,	Slot<>0	–	**arbitrate**
reply	Slot=0	backscatter new RN16	**reply**
acknowledged, open, secured	all	transition **inventoried** from *A*→*B* or *B*→*A*	**battery ready**
killed	all	–	**killed**

[a] *QueryAdjust*, in the arbitrate or reply states, instructs a Tag to load a new random value into its slot counter.

[b] In Manchester with Long Activation and Interrogator locking in effect, the Tag checks first the Interrogator ID, if it matches, it proceeds with the command execution, otherwise, it ignores the command.

C.3.4 Command response: *ACK*

Table C.39 — *ACK* command-response table [a]

Starting State	Condition	Response	Next State
battery ready	All	–	**battery ready**
arbitrate	All	–	**arbitrate**
reply	valid RN16; no Simple Sensor functionality	backscatter {PC, XPC_W1(if XI=1), XPC_W2(if XEB=1), UII, PacketCRC} or {00000_2, truncated UII, StoredCRC}	**acknowledged**
	valid RN16; Simple Sensor functionality (SS Resp =1)	backscatter {PC, XPC_W1, XPC_W2(if XEB=1), UII, SSD, PacketCRC} or {00000_2, truncated UII, StoredCRC}	**acknowledged**
	valid RN16; Mobile RFID(XPC_W1 212h=1)	backscatter {PC, XPC_W1, XPC_W2(if XEB=1), UII, MIIM content name, PacketCRC} or {000002, truncated UII, StoredCRC}	**acknowledged**
	invalid RN16	–	**reply**
acknowledged	valid RN16; no Simple Sensor functionality	backscatter {PC, XPC_W1(if XI=1), XPC_W2(if XEB=1), UII, PacketCRC} or {00000_2, truncated UII, StoredCRC}	**acknowledged**
	valid RN16; Simple Sensor functionality (SS Resp =1)	backscatter {PC, XPC_W1, XPC_W2(if XEB=1), UII, SSD, PacketCRC} or {00000_2, truncated UII, StoredCRC}	**acknowledged**
	valid RN16; Mobile RFID(XPC_W1 212h=1)	backscatter {PC, XPC_W1, XPC_W2(if XEB=1), UII, MIIM content name, PacketCRC} or {000002, truncated UII, StoredCRC}	**acknowledged**
	invalid RN16	–	**arbitrate**

[a] See *Query_BAT* command definitions for more details when there is a SSD associatted with the response.

Table C.39 *(continued)*

Starting State	Condition	Response	Next State
open	valid handle; no Simple Sensor functionality	backscatter { PC, XPC_W1(if XI=1), XPC_W2(if XEB=1), UII, PacketCRC} or {00000_2, truncated UII, StoredCRC}	**open**
	valid handle; Simple Sensor functionality (SS Resp =1)	backscatter {PC, XPC_W1, XPC_W2(if XEB=1), UII, SSD, PacketCRC} or {00000_2, truncated UII, StoredCRC}	**open**
	invalid handle	–	**open**
secured	valid handle; no Simple Sensor functionality	backscatter { PC, XPC_W1(if XI=1), XPC_W2(if XEB=1), UII, PacketCRC} or {00000_2, truncated UII, StoredCRC}	**secured**
	valid handle; Simple Sensor functionality (SS Resp =1)	backscatter {PC, XPC_W1, XPC_W2(if XEB=1), UII, SSD, PacketCRC} or {00000_2, truncated UII, StoredCRC}	**secured**
	invalid handle	–	**secured**
killed	all		**killed**

[a] See *Query_BAT* command definitions for more details when there is a SSD associatted with the response.

C.3.5 Command response: *NAK*

Table C.40 — *NAK*[b] command-response table [a]

Starting State	Condition	Response	Next State
battery ready	all	–	**battery ready**
arbitrate, reply, acknowledged, open, secured	all	–	**arbitrate**
killed	all	–	**killed**

[a] In Manchester with Long Activation and Interrogator locking in effect, the Tag checks first the Interrogator ID, if it matches, it proceeds with the command execution, otherwise, it ignores the command.

[b] Notice this command contains session as a parameter.

C.3.6 Command response: *Req_RN*

See C.1.7.

C.3.7 Command response: *Select*

Table C.41 — *Select* command-response table [a,b]

Starting State	Condition	Response	Next State
hibernate	all	–	**hibernate**
stateful hibernate	all	–	**stateful hibernate**
battery ready, arbitrate, reply, acknowledged, open, secured	all	assert or de-assert **SL**, or set **inventoried** to *A* or *B*	**battery ready**
killed	all	–	**killed**

[a] If session locking is in effect, the Tag only accepts the command if the session matches the activating session.

[b] If Interrogator locking in effect, the Tag first checks the Interrogator ID, if it matches, it proceeds with the command execution; if Interrogator locking is in effect but it does not match, it ignores the command; if Short Activation, it proceeds with the command execution.

C.3.8 Command response: *Read*

See C.1.9.

C.3.9 Command response: *Write*

See C.1.10.

C.3.10 Command response: *Kill*

See C.1.11.

C.3.11 Command response: *Lock*

See C.1.12.

C.3.12 Command response: *Access*

See C.1.13.

C.3.13 Command response: *BlockWrite*

See C.1.14.

C.3.14 Command response: *BlockErase*

See C.1.15.

C.3.15 Command response: *BlockPermalock*

See C.1.16

C.3.16 Command response: *Challenge*

Table C.42 — *Challenge* command-response table [a]

Starting State	Condition	Response	Next State
hibernate	all	–	**hibernate**
stateful hibernate	all	–	**stateful hibernate**
battery ready, arbitrate, reply, acknowledged, open, secured	supported security timeout, unsupported CSI, not-executable message, nonzero RFU bits	set **C**=0	**battery ready**
	supported CSI and executable message	store result, set **C**=1	**battery ready**
killed	all	–	**killed**

[a] If Interrogator locking in effect, the Tag first checks the Interrogator ID, if it matches, it proceeds with the command execution; if Interrogator locking is in effect but it does not match, it ignores the command; if Short Activation, it proceeds with the command execution.

C.3.17 Command response: *Authenticate*

See C.1.18

C.3.18 Command response: *AuthComm*

See C.1.19.

C.3.19 Command response: *SecureComm*

See C.1.20.

C.3.20 Command response: *ReadBugger*

See C.1.21.

C.3.21 Command response: *KeyUpdate*

See C.1.22.

C.3.22 Command response: *Untraceable*

See C.1.23.

C.3.23 Command response: *FileSetup*

See C.1.24.

C.3.24 Command response: *FileOpen*

See C.1.25.

C.3.25 Command response: *FilePrivilege*

See C.1.26.

C.3.26 Command response: *TagPrivilege*

See C.1.27.

C.3.27 Command response: *FileList*

See C.1.28.

C.3.28 Command response: T_2 timeout

See C.1.29.

C.3.29 Command response: *Long Activation*

Table C.43 — *Long Activation* command-response table

Starting State	Condition	Response	Next State
hibernate, stateful hibernate	Valid activation preamble detected	–	**activation code check**
	Otherwise	–	**hibernate, stateful hibernate** [a]
activation code check	Wildcard or matching Activation Mask detected	If Session Locking On, program specified session timeout timer. If Session Locking Off, clear all **inventoried** flag timeout timers	**battery ready**
	Otherwise	–	**hibernate, stateful hibernate** [a]
battery ready, arbitrate, reply, acknowledged, open, secured	Commanded to receive data rate mode of 16 Kbps or higher	Ignore command	**battery ready, arbitrate, reply, acknowledged, open, secured**
	Commanded to receive data rate mode of 8 Kbps	Manufacturer option to ignore or "re-activate". If re-activation supported, then set **inventoried** flag to *A* for the active battery ready session, deassert **SL**, and go to **hibernate** mode/**activation code check** state	**activation code check**
killed	Valid activation preamble detected	–	**killed**

[a] Return to **stateful hibernate** if at least one **inventoried** flag timer is still running.

C.3.30 Command response: *Short Activation*

Table C.44 — *Short Activation* command-response table

Starting State	Condition	Response	Next State
hibernate, stateful hibernate	valid activation preamble detected	–	**activation code check**
	otherwise	–	**hibernate, stateful hibernate** [a]

[a] Return to **stateful hibernate** if at least one **inventoried** flag timer is still running.

Table C.44 *(continued)*

Starting State	Condition	Response	Next State
activation code check	Wildcard or matching Activation Mask detected	Clear all timers, set **inventoried** flags to A, deassert **SL**	**battery ready**
	otherwise	–	**hibernate, stateful hibernate** [a]
battery ready, arbitrate, reply, acknowledged, open, secured	Commanded to receive data rate mode of 16 Kbps or higher	Ignore command	**battery ready, arbitrate, reply, acknowledged, open, secured**
	Commanded to receive data rate mode of 8 Kbps	Manufacturer option to ignore or "re-activate". If re-activation supported, then set **inventoried** flag to *A* for the active battery ready session, deassert **SL**, and go to **hibernate** mode/**activation code check** state	**activation code check**
killed	valid activation preamble detected	–	**killed**

[a] Return to **stateful hibernate** if at least one **inventoried** flag timer is still running.

C.3.31 Command response: *Query_BAT*

Table C.45 — *Query_BAT* [a,d] **command-response table**

Starting State	Condition	Response	Next State
hibernate	all	–	**hibernate**
stateful hibernate	all	–	**stateful hibernate**
battery ready, arbitrate, reply [c,d]	Tag operating in battery assisted mode; slot=0; matching **Tag type** & **inventoried** [b] & **SL** flags	backscatter new RN16	**reply**
	slot<>0; matching **Tag type & inventoried** [b] & **SL** flags	–	**arbitrate**
	otherwise	–	**battery ready**

[a] *Query_BAT* (in any state other than killed) starts a new round and may change the session (if session locking is off); *Query_BAT* also instructs a Tag to load a new random value into its slot counter.

[b] As described in 6.3.2.10, a Tag transitions its **inventoried** flag prior to evaluating the condition.

[c] If session locking is in effect, the Tag only accepts the command if the session matches the activating session.

[d] If Interrogator locking in effect, the Tag first checks the Interrogator ID, if it matches, it proceeds with the command execution; if Interrogator locking is in effect but it does not match, it ignores the command; if Short Activation, it proceeds with the command execution.

[e] Tag checks the Tag Type Select field, if it matches the conditions in that field, Tag executes this command, otherwise, it ignores it.

Table C.45 *(continued)*

Starting State	Condition	Response	Next State
acknowledged, open, secured [c,d]	Tag operating in battery assisted mode; slot=0; matching **Tag type & inventoried** [b] & **SL** flags	backscatter new RN16; transition **inventoried** [b] from *A*→*B* or *B*→*A* if and only if new session matches prior session	**reply**
	Tag operating in battery assisted mode; slot<>0; matching **Tag type & inventoried** [b] & **SL** flags	transition **inventoried** [b] from *A*→*B* or *B*→*A* if and only if new session matches prior session	**arbitrate**
	Tag operating in battery assisted mode; all other conditions	transition **inventoried** [b] from *A*→*B* or *B*→*A* if and only if new session matches prior session	**battery ready**
killed	all	–	**killed**

[a] *Query_BAT* (in any state other than killed) starts a new round and may change the session (if session locking is off); *Query_BAT* also instructs a Tag to load a new random value into its slot counter.

[b] As described in 6.3.2.10, a Tag transitions its **inventoried** flag prior to evaluating the condition.

[c] If session locking is in effect, the Tag only accepts the command if the session matches the activating session.

[d] If Interrogator locking in effect, the Tag first checks the Interrogator ID, if it matches, it proceeds with the command execution; if Interrogator locking is in effect but it does not match, it ignores the command; if Short Activation, it proceeds with the command execution.

[e] Tag checks the Tag Type Select field, if it matches the conditions in that field, Tag executes this command, otherwise, it ignores it.

C.3.32 Command response: *Next*

Table C.46 — *Next* command-response table

Starting State	Condition	Response	Next State
hibernate,	all	–	**hibernate**
stateful hibernate	all	–	**stateful hibernate**
battery ready	all	–	**battery ready**
arbitrate, reply	all	-	**arbitrate, reply**
acknowledged, open, secured	valid handle, session **hibernate** timer running	backscatter RN16 handle or RN16, set sensitivity as indicated if that sensitivity is supported	**stateful hibernate**
	valid handle, no session **hibernate** timer	backscatter RN16 handle or RN16, set sensitivity as indicated if that sensitivity is supported	**hibernate**
	otherwise	–	**acknowledged, open, secured**
killed	all	–	**killed**

[a] Tags may switch to passive mode in the course of an inventory round if battery is drained below critical threshold; In passive mode *Next* is no longer supported.

C.3.33 Command response: *Deactivate_BAT*

Table C.47 — *Deactivate_BAT* [a] command-response table

Starting State	Condition	Response	Next State
battery ready, arbitrate, reply, acknowledged, open, secured	Matching Interrogator ID (if Interrogator locking is in effect), matching activating session (if session locking is in effect), matching **inventoried** & **SL** flags, Override=0	–	**stateful hibernate**
	Matching Interrogator ID (if Interrogator locking is in effect), Override=1	Set **inventoried** flags to A, deassert **SL**, and clear timers	**stateful hibernate**
	Matching Interrogator ID (if Interrogator locking is in effect), matching activating session (if session locking is in effect), matching **inventoried** but mismatching **SL**, Override=0	–	**battery ready**
	Matching Interrogator ID (if Interrogator locking is in effect), matching activating session (if session locking is in effect), mismatching **inventoried**, **SL** does not matter, Override=0	Set **inventoried** flag to A	**battery ready**
	Mismatching Interrogator ID (if Interrogator locking is in effect) or mismatching activating session (if session locking is in effect), Override=0	–	**battery ready ,arbitrate, reply, acknowledged, open, secured**
killed	all	–	**killed**

[a] Don't care' in **inventoried** flag use field is interpreted as any flag state matches.

C.3.34 Command response: *Broadcast ID*

Table C.48 — *Broadcast ID* command-response table

Starting State	Condition	Response	Next State
hibernate,	all	–	**hibernate**
stateful hibernate	all	–	**stateful hibernate**
battery ready, arbitrate, reply, acknowledged, open, secured	all	–	**battery ready, arbitrate, reply, acknowledged, open, secured**
killed	all	–	**killed**

C.3.35 Command response: *Multirate_Reset*

Table C.49 — *Multirate_Reset* command-response table

Starting State	Condition	Response	Next State
hibernate,	All	–	**hibernate**
stateful hibernate	All	–	**stateful hibernate**
battery ready, arbitrate, reply, acknowledged, open, secured	All	Clear all timers, set **inventoried** flags to A, deassert **SL**. This applies to all Normal Mode (not Hibernation, see Figure 80) states, including intermediate steps within the *Kill* or *Access* sequences prior to the Tag completing the sequence (receiving both halves of the appropriate password).	**hibernate**
killed	All	–	**killed**

C.3.36 Command response: *HandleSensor*

Table C.50 — *HandleSensor* command-response table

Starting State	Condition	Response	Next State
hibernate, stateful hibernate	All	–	**hibernate, stateful hibernate** [b]
activation code check	All	–	**activation code check**
battery ready	All	–	**battery ready**
arbitrate, reply, acknowledged	all	–	**arbitrate**

[a] Sensor specific output of HandleSensor command payload processing.

[b] Return to **stateful hibernate** if at least one **inventoried** flag timer is still running.

Table C.50 *(continued)*

Starting State	Condition	Response	Next State
open, secured	valid handle & valid command payload	backscatter header = 0, response code [a] and handle when internal processing done	**open, secured**
	valid handle & invalid command payload	backscatter header = 1, error code (see Annex I) and handle	**open, secured**
	invalid handle	–	**open, secured**
killed	all	–	**killed**

[a] Sensor specific output of HandleSensor command payload processing.

[b] Return to **stateful hibernate** if at least one **inventoried** flag timer is still running.

C.3.37 Command response: *BroadcastSync*

See C.2.5.

C.3.38 Command response: Session Flag timer timeout

Table C.51 — Session Flag timer timeout command-response table

Starting State	Condition	Response	Next State
stateful hibernate	all	Set that flag to A	**stateful hibernate**
activation code check	all	Set that flag to *A* as soon as current operations allow	**activation code check**
battery ready	all	Set that flag to *A* as soon as current operations allow	**battery ready**
arbitrate	all	Set that flag to *A* as soon as current operations allow	**arbitrate**
reply	all	Set that flag to *A* as soon as current operations allow	**reply**
acknowledged	all	Set that flag to *A* as soon as current operations allow	**acknowledged**
open	all	Set that flag to *A* as soon as current operations allow	**open**
secured	all	Set that flag to *A* as soon as current operations allow	**secured**
killed	all	–	**killed**

C.3.39 Command response: INACT_T or Selective Global Timeout

Table C.52 — INACT_T [a] or Selective Global Timeout command-response table

Starting State	Condition	Response	Next State
battery ready, arbitrate, open, secured	all	Set active **inventoried** flag to *A* and clear active session flag timeout timer, **SL→~SL**	**stateful hibernate**
killed	all	–	**killed**

[a] See clause 7.3.2.2 for details on the refresh of the INACT_T timer

C.3.40 Command response: Global Timeout

Table C.53 — Global Timeout command-response table

Starting State	Condition	Response	Next State
battery ready, arbitrate, reply, acknowledged, open, secured	all	Set active **inventoried** flag to *A* and clear active session flag timeout timer, **SL→~SL**	**stateful hibernate**
killed	all	–	**killed**

C.3.41 Command response: T_A

Table C.54 — T_A command-response table

Starting State	Condition	Response	Next State
activation code check	all	–	**battery ready**
killed	all	–	**killed**

C.3.42 Command response: OpRegister Read/Write

Table C.55 — *OpRegister Read/Write* command-response table

Starting State	Condition	Response	Next State
hibernate	all	–	**hibernate**
stateful hibernate	all	–	**stateful hibernate**
battery ready	all	–	**battery ready**
arbitrate, reply, acknowledged	all	–	**arbitrate**
open	all	–	**open**
secured	valid handle, read, valid reg ID and wordcount	Successful response is to backscatter header (value = “0”) bit, data being read, RN handle, and CRC-16 as per Table 7.33	**secured**
	valid handle, write, valid reg ID and wordcount	Successful response is to backscatter header (value = “0”) bit, RN handle, and CRC-16 as per Table 7.33	**secured**
	valid handle, invalid reg ID, wordcount, or any other failure	Failure response is to backscatter header (value = “1”) bit, error code as per Annex I, RN handle, and CRC-16.	**secured**
	invalid handle	–	**secured**
killed	all	–	**killed**

C.3.43 Command response: Invalid command

Table C.56 — Invalid command table

Starting State	Condition	Response	Next State
battery ready [a]	all	–	**battery ready**
arbitrate [b]	all	–	**arbitrate**
reply [b]	all	–	**reply**
acknowledged [b]	all	–	**acknowledged**
open [b]	all, excluding valid commands interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively (see Figure 6.24 and Figure 6.26).	–	**open**
	otherwise valid commands, except *Req_RN*, *Query_BAT* interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively (see Figure 6.24 and Figure 6.26).	–	**arbitrate**
secured [b]	all, excluding valid commands interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively (see Figure 6.24 and Figure 6.26).	–	**secured**
	otherwise valid commands, except *Req_RN*, *Query*, *Query_BAT* interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively (see Figure 6.24 and Figure 6.26).	–	**arbitrate**
killed [a]	All	–	**killed**

[a] "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, or any other command either not recognized or not executable by the Tag.

[b] "Invalid" shall mean an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than a *Query_BAT with session locking off*) with a session parameter not matching that of the inventory round currently in progress, an otherwise valid command interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively; or any other command either not recognized or not executable by the Tag.