
**Information technology — Radio
frequency identification (RFID) for item
management: Data protocol —**

**Part 1:
Application interface**

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

Partie 1: Interface d'application



Reference number
ISO/IEC 15961-1:2013(E)

AVA Documents
Document Sharing Hub
© ISO/IEC 2013



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2013

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

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

Published in Switzerland

Contents

Page

Foreword	viii
Introduction.....	ix
1 Scope	1
2 Normative references	1
3 Terms, definitions and conventions	2
3.1 Terms and definitions	2
3.2 Conventions	2
4 Compliance	2
4.1 General	2
4.2 Application compliance	3
4.3 Conformance of the Data Processor	3
5 Protocol model	3
5.1 Overview.....	3
5.2 Layered protocol	3
5.2.1 Application layer - as defined in the various parts of ISO/IEC 15961	4
5.2.2 Application interface - as defined in ISO/IEC 15961-1	5
5.2.3 Data Protocol Processing - as defined in ISO/IEC 15962	5
5.2.4 Data Protocol Interface - as defined in ISO/IEC 15962	5
5.3 Flexible implementation configurations	6
5.4 Functional processes – interrogator implementation	6
5.4.1 Functional processes - application interface	7
5.4.2 Functional processes – interrogator	7
5.4.3 RFID tag	8
5.5 ISO/IEC 15962 and the Data Processor	9
6 Presentation conventions.....	9
6.1 Presentation of commands, responses and arguments	9
6.1.1 Commands and responses	9
6.1.2 Arguments.....	9
6.1.3 Data types	10
6.2 Object Identifier presentation in the application interface.....	10
6.2.1 Object identifier structure to ISO/IEC 8824-1.....	10
6.2.2 Presenting the Object-Identifier in the style of ISO/IEC 8824-1	11
6.2.3 Presenting the Object-Identifier as a Uniform Resource Name (URN)	12
6.3 Byte Notation	12
6.3.1 The byte: the basic unit for 8-bit coding	12
6.3.2 Bit ordering	12
6.3.3 Byte conversion.....	12
7 Processing application commands and responses	12
7.1 General	12
7.2 Encoding system related information in commands.....	13
7.2.1 Singulation-Id.....	13
7.2.2 AFI	13
7.2.3 DSFID	14
7.2.4 Access-Method	14
7.2.5 Data-Format	16
7.3 Preparing the basic Objects and other application-based arguments	17
7.3.1 General model.....	17
7.3.2 Object-Identifier	17

7.3.3	Relating Object-Identifiers	18
7.3.4	Object.....	18
7.3.5	Compact-Parameter.....	18
7.3.6	Object-Lock	20
7.4	Other command arguments.....	20
7.4.1	Access-Password	20
7.4.2	Additional-App-Bits	21
7.4.3	AFI-Lock.....	21
7.4.4	Append-To-Existing-Multiple-Record	21
7.4.5	Application-Defined-Record-Capacity	21
7.4.6	Avoid-Duplicate.....	21
7.4.7	Battery-Assist-Indicator	21
7.4.8	Block-Align	21
7.4.9	Block-Align-Packed-Object.....	21
7.4.10	Check-Duplicate.....	22
7.4.11	Data-CRC-Indicator.....	22
7.4.12	Data-Length-Of-Record	22
7.4.13	Delete-MR-Method	22
7.4.14	Directory-Length-EBV8-Indicator.....	22
7.4.15	DSFID-Lock.....	22
7.4.16	DSFID-Pad-Bytes	23
7.4.17	Editable-Pointer-Size	23
7.4.18	Encoded-Memory-Capacity	23
7.4.19	EPC-Code	23
7.4.20	Full-Function-Sensor-Indicator	23
7.4.21	Hierarchical-Identifier-Arc.....	23
7.4.22	Identifier-Of-My-Parent.....	23
7.4.23	Identify-Method	23
7.4.24	ID-Type.....	24
7.4.25	Instance-Of-Arc.....	24
7.4.26	Kill-Password	24
7.4.27	Length-Of-Mask.....	24
7.4.28	Lock-Directory-Entry	24
7.4.29	Lock-Multiple-Records-Header.....	24
7.4.30	Lock-Record-Preamble	25
7.4.31	Lock-UII-Segment-Arguments	25
7.4.32	Max-App-Length.....	25
7.4.33	Memory-Bank	25
7.4.34	Memory-Bank-Lock.....	25
7.4.35	Memory-Length-Encoding	25
7.4.36	Memory-Segment.....	25
7.4.37	Memory-Type.....	25
7.4.38	Multiple-Records-Directory-Length	25
7.4.39	Multiple-Records-Features-Indicator	26
7.4.40	NSI-Bits	26
7.4.41	Number-In-Data-Element-List.....	26
7.4.42	Number-Of-Records	26
7.4.43	Number-Of-Tags.....	26
7.4.44	Objects-Offsets-Multiplier	27
7.4.45	Packed-Object-Directory-Type	27
7.4.46	Password	27
7.4.47	Password-Type	27
7.4.48	PO-Directory-Size	27
7.4.49	PO-Index-Length.....	27
7.4.50	Pointer.....	27
7.4.51	Pointer-To-Multiple-Records-Directory	27
7.4.52	Read-Record-Type	28
7.4.53	Read-Type.....	29
7.4.54	Record-Memory-Capacity	30
7.4.55	Record-Type-Arc.....	30

7.4.56	Record-Type-Classification	30
7.4.57	Sector-Identifier	30
7.4.58	Simple-Sensor-Indicator	31
7.4.59	Start-Address-Of-Record	31
7.4.60	Tag-Data-Profile-ID-Table	31
7.4.61	Tag-Mask	31
7.4.62	Update-Multiple-Records-Directory	31
7.4.63	Word-Count	31
7.4.64	Word-Pointer	31
7.5	Command-related field names	31
7.5.1	Data-Set	32
7.5.2	Identities	32
7.5.3	Length-Lock Byte	32
7.5.4	Length-Of-Encoded-Data	32
7.5.5	Lock-Status	32
7.5.6	Logical-Memory-Map	32
7.5.7	Memory-Capacity	32
7.5.8	Module-OID	32
7.5.9	Number-Of-Tags-Found	32
7.5.10	PO-ID-Table	32
7.5.11	Protocol-Control-Word	32
7.5.12	Read-Data	33
7.6	Data security	33
8	Data flows and processes to the air interface	33
8.1	Establishing communications between the application and the RFID tag	33
8.1.1	Air interface services	33
8.1.2	System information	34
8.2	Application system services	34
9	Command-Codes, Completion-Codes, and Execution-Codes	34
9.1	Final arc values of the command and response modules	35
9.2	Completion-Code	36
9.3	Execution-Code	39
10	Commands and responses	40
10.1	Configure-AFI	40
10.1.1	Configure-AFI command	40
10.1.2	Configure-AFI response	40
10.2	Configure-DSFID	41
10.2.1	Configure-DSFID command	41
10.2.2	Configure-DSFID response	42
10.3	Inventory-Tags	42
10.3.1	Inventory-Tags command	42
10.3.2	Inventory-Tags response	44
10.4	Delete-Object	44
10.4.1	Delete-Object command	44
10.4.2	Delete-Object response	45
10.5	Modify-Object	46
10.5.1	Modify-Object command	46
10.5.2	Modify-Object response	47
10.6	Read-Object-Identifiers	48
10.6.1	Read-Object-Identifiers command	48
10.6.2	Read-Object-Identifiers response	49
10.7	Read-Logical-Memory-Map	49
10.7.1	Read-Logical-Memory-Map command	49
10.7.2	Read-Logical-Memory-Map response	50
10.8	Erase-Memory	50
10.8.1	Erase-Memory command	50
10.8.2	Erase-Memory response	51
10.9	Get-App-Based-System-Info	51

10.9.1	Get-App-Based-System-Info command.....	51
10.9.2	Get-App-Based-System-Info response.....	51
10.10	Write-Objects.....	52
10.10.1	Write-Objects command	52
10.10.2	Write-Objects response	54
10.11	Read-Objects.....	55
10.11.1	Read-Objects command.....	55
10.11.2	Read-Objects response.....	56
10.12	Write-Objects-Segmented-Memory-Tag	56
10.12.1	Write-Objects-Segmented-Memory-Tag command	56
10.12.2	Write-Objects-Segmented-Memory-Tag response	58
10.13	Write-EPC-Ull.....	59
10.13.1	Write-EPC-Ull command	59
10.13.2	Write-EPC-Ull response	60
10.14	Inventory-ISO-Ullmemory	60
10.14.1	Inventory-ISO-Ullmemory command	60
10.14.2	Inventory-ISO-Ullmemory response	61
10.15	Inventory-EPC-Ullmemory	62
10.15.1	Inventory-EPC-Ullmemory command	62
10.15.2	Inventory-EPC-Ullmemory response	62
10.16	Write-Password-Segmented-Memory-Tag	63
10.16.1	Write-Password-Segmented-Memory-Tag command	63
10.16.2	Write-Password-Segmented-Memory-Tag response	63
10.17	Read-Words-Segmented-Memory-Tag	64
10.17.1	Read-Words-Segmented-Memory-Tag command	64
10.17.2	Read-Words-Segmented-Memory-Tag response	64
10.18	Kill-Segmented-Memory-Tag	65
10.18.1	Kill-Segmented-Memory-Tag command	65
10.18.2	Kill-Segmented-Memory-Tag response	65
10.19	Delete-Packed-Object.....	65
10.19.1	Delete-Packed-Object command.....	65
10.19.2	Delete-Packed-Object response.....	66
10.20	Modify-Packed-Object-Structure.....	67
10.20.1	Modify-Packed-Object-Structure command.....	67
10.20.2	Modify-Packed-Object-Structure response.....	68
10.21	Write-Segments-6TypeD-Tag.....	69
10.21.1	Write-Segments-6TypeD-Tag command	69
10.21.2	Write-Segments-6TypeD-Tag response	70
10.22	Read-Segments-6TypeD-Tag.....	71
10.22.1	Read-Segments-6TypeD-Tag command.....	71
10.22.2	Read-Segments-6TypeD-Tag response.....	73
10.23	Write-Monomorphic-Ull	73
10.23.1	Write-Monomorphic-Ull command	73
10.23.2	Write-Monomorphic-Ull response	75
10.24	Configure-Extended-DSFID	76
10.24.1	Configure-Extended-DSFID command	76
10.24.2	Configure-Extended-DSFID response	77
10.25	Configure-Multiple-Records-Header	77
10.25.1	Configure-Multiple-Records-Header command	77
10.25.2	Configure-Multiple-Records-Header response	80
10.26	Read-Multiple-Records.....	80
10.26.1	Read-Multiple-Records command.....	80
10.26.2	Read-Multiple-Records response.....	81
10.27	Delete-Multiple-Record.....	82
10.27.1	Delete-Multiple-Record command.....	82
10.27.2	Delete-Multiple-Record response.....	83
11	Arguments	84
11.1	Add-Objects.....	84
11.2	DSFID-Constructs	85

11.3	EPC-Ullmemory	85
11.4	Ext-DSFID-Constructs	85
11.5	ISO-Ullmemory	87
11.6	Item-Related-Add-Objects	87
11.7	Item-Related-DSFID-Constructs	87
11.8	Multiple-Records-Constructs	87
11.9	Multiple-Records-Directory-Structure	89
11.10	Multiple-Records-Header-Structure	89
11.11	Multiple-Records-Preamble-Structure	90
11.12	Packed-Object-Constructs	91
11.13	Read-Objects	93
11.14	Read-Objects-Response	93
11.15	Read-oids-Response	93
11.16	Ull-Add-Objects	93
11.17	Ull-DSFID-Constructs	94
11.18	Write-Responses	94
Annex A (informative) Abstract syntax and transfer encoding rules		95
Annex B (informative) Accommodating established data formats		104
Annex C (informative) Relating data Objects		106
Annex D (informative) Data security issues		108
Annex E (informative) Original commands and responses using ASN.1 abstract syntax		110
Annex F (informative) Example of a transfer encoding to ISO/IEC 15961:2004		139
Bibliography		142

Foreword

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

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

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

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.

ISO/IEC 15961-1 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 31, *Automatic identification and data capture techniques*.

This first edition of ISO/IEC 15961-1, together with ISO/IEC 15961-2, ISO/IEC 15961-3 and ISO/IEC 15961-4, cancels and replaces ISO/IEC 15961:2004, which has been technically revised.

ISO/IEC 15961 consists of the following parts, under the general title *Information technology — Radio frequency identification (RFID) for item management: Data protocol*:

- *Part 1: Application interface*
- *Part 2: Registration of RFID data constructs*
- *Part 3: RFID data constructs*

The following part is under preparation:

- *Part 4: Application interface commands for battery assist and sensor functionality*

Introduction

The technology of radio frequency identification (RFID) is based on non-contact electronic communication across an air interface. The structure of the bits stored on the memory of the RFID tag is invisible and accessible between the RFID tag and the interrogator only by the use of an air interface protocol, as specified in the appropriate part of ISO/IEC 18000. The result of the transfer of data between an application and an interrogator in open systems requires data to be encoded in a consistent manner on any RFID tag that is part of that open system. This is not only to allow equipment to be interoperable, but in the special case of data carriers, for the data to be encoded on the RFID tag in one systems implementation and to be read at a later time in a completely different and unknown systems implementation. The data bits stored on each RFID tag must be formatted in such a way as to be reliably read at the point of use if the RFID tag is to fulfil its basic objective. This reliability is achieved through the specification of a data protocol in this part of ISO/IEC 15961 and the data encoding rules of ISO/IEC 15962. Additionally, ISO/IEC 24791-1 specifies a software system infrastructure architecture that enables RFID system operations between business applications and RFID interrogators. Specific parts of the infrastructure standards address data management requirements (ISO/IEC 24791-2) and device interface requirements (ISO/IEC 24791-5). These support defined implementations that incorporate the encoding rules of ISO/IEC 15962 and the functional rules of the commands and responses in this part of ISO/IEC 15961.

Manufacturers of RFID equipment (interrogators, RFID tags, etc.) and users of RFID technology require standards-based data protocols for RFID for item management. This part of ISO/IEC 15961, ISO/IEC 15962, and ISO/IEC 24791 specify these protocols, which are layered above the air interface standards defined in ISO/IEC 18000.

The transfer of data to and from an application, supported by appropriate application commands, is the subject of this part of ISO/IEC 15961. The companion International Standard, ISO/IEC 15962, specifies the overall process and the methodologies developed to format the application data into a structure to store on the RFID tag.

Information technology — Radio frequency identification (RFID) for item management: Data protocol —

Part 1: Application interface

1 Scope

This part of ISO/IEC 15961 focuses on the abstract interface between an application and the data processor, and includes the specification and definition of application commands and responses. It allows data and commands to be specified in a standardised way, independent of the particular air interface of ISO/IEC 18000.

This part of ISO/IEC 15961

- provides guidelines on how data shall be presented as objects;
- defines the structure of Object Identifiers, based on ISO/IEC 9834-1;
- specifies the commands that are supported for transferring data between an application and the radio frequency identification (RFID) tag;
- specifies the responses that are supported for transferring data between the RFID tag and the application;
- does not specify any required transfer syntax with ISO/IEC 15962, but provides the non-normative information in Annex A to provide backward compatibility with ISO/IEC 15961:2004.

It is expected that this part of ISO/IEC 15961 will be used as a reference to develop software appropriate for particular applications, or for particular RFID equipment.

2 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 9834-1, *Information technology — Open Systems Interconnection — Procedures for the operation of OSI Registration Authorities: General procedures and top arcs of the International Object Identifier tree* (equivalent to ITU-T Recommendation X.660)

ISO/IEC 15961-3, *Information technology — Radio frequency identification (RFID) for item management: Data protocol — Part 3: RFID data constructs*

ISO/IEC 15962:2013, *Information technology — Radio frequency identification (RFID) for item management — Data protocol: data encoding rules and logical memory functions*

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

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

3 Terms, definitions and conventions

3.1 Terms and definitions

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

3.1.1

application

software component that issues commands and receives responses to the commands within a system

3.1.2

Data Processor

implementation of the processes defined in ISO/IEC 15962, including the Data Compactor, Formatter, Logical Memory, and Command/Response Unit

NOTE This was called the Data Protocol Processor in ISO/IEC 15691:2004.

3.1.3

Relative-OID

particular Object Identifier where a common Root-OID (for the first and subsequent arcs) is implied, and remaining arcs after the Root-OID are defined by the Relative-OID

3.2 Conventions

Conventionally in International Standards, long numbers are separated by a space character as a “thousands separator”. This convention has not been followed in this part of ISO/IEC 15961 because the arcs of an Object Identifier are defined by a space separator (according to ISO/IEC 8824 and ISO/IEC 8825). As the correct representation of these arcs is vital to this part of ISO/IEC 15961, all numeric values have no space separators except to denote a node between two arcs of an Object Identifier.

4 Compliance

4.1 General

The commands and responses in this part of ISO/IEC 15961 are only expressed in an abstract syntax, and transfer encoding is no longer required. As such, compliance to this part of ISO/IEC 15961 for a particular system is specifically indicated by the resultant proper encoding of RFID tags according to ISO/IEC 15962 by the system.

The arguments and fields contained in individual commands and responses identify what needs to be taken into account for correct input to the Data Processor to achieve a valid encoding. Also, they identify what an application expects to have returned following access to an RFID tag. Because of the way the Data Protocol is structured, the commands and responses specified in this part of ISO/IEC 15961 are, to a large extent, independent of particular RFID tag types that are only known to the Data Processor through the Tag Driver. The effect of this is that ISO/IEC 15962 can specify conformance requirements for valid encoding, which this part of ISO/IEC 15961 cannot.

The following sub-clauses provide compliance advice as best practice to achieve an integrated data communication channel between the application and the RFID tag.

4.2 Application compliance

An application is expected to support the commands and responses that are meaningful to the application. For every command considered relevant for an application, all the constituent components need to be taken into account in transfers between the application and the Data Processor.

In particular, application standards need to take into consideration the various arguments in the command as defined in Clause 7 (e.g. **Object-Lock**, **Compact-Parameter**). These determine the requirements of what is encoded on the RFID tag, and the necessary processes that the Data Processor has to invoke to achieve a valid encoding.

4.3 Conformance of the Data Processor

The Data Processor is, effectively, the implementation of ISO/IEC 15962. Depending on the scope of the Data Processor (ranging from being specific to an industry to being generic to the entire RFID Data Protocol) various arguments included in the commands can be processed in different manners (e.g. data can be identified with a full **Object-Identifier** or a **Relative-OID**). This part of ISO/IEC 15961 imposes no constraints on the design of the Data Processor, other than a requirement to support all the functionality specified by the arguments in the commands that are necessary to achieve proper encoding.

5 Protocol model

5.1 Overview

RFID supports bit encodation in the RFID tag memory. Unlike other data carrier standards prepared by ISO/IEC JTC1 SC31 which require encodation schemes that are specific to the individual data carrier technology, ISO/IEC 18000 does not specify the interpretation of bits or bytes encoded on the RFID tag memory. However, as an RFID tag is a relay in a communication system, each tag used for open systems item management needs to have data encoded in a consistent manner. The prime function of this part of ISO/IEC 15961 is to specify a common interface between the application programs and the RFID interrogator. The prime function of ISO/IEC 15962 is to specify the common encoding rules and logical memory functions.

RFID tags utilise electronic memory, which is typically capable of increasing data capacity as new generations of product are introduced. Differences in data capacity of each RFID tag type, whether similar or dissimilar, are recognised by the data protocol defined in these two International Standards.

Different application standards may have their own particular data sets or data dictionaries. Each major application standard for item management needs to have its data treated in an unambiguous manner, avoiding confusion with data from other applications and even with data from closed systems. The data protocol specified in these International Standards ensures the unambiguous identification of data.

5.2 Layered protocol

The protocol layers of an implementation of RFID for item management are illustrated schematically in Figure 1 — Schematic of protocol layers for an implementation of RFID for item management.

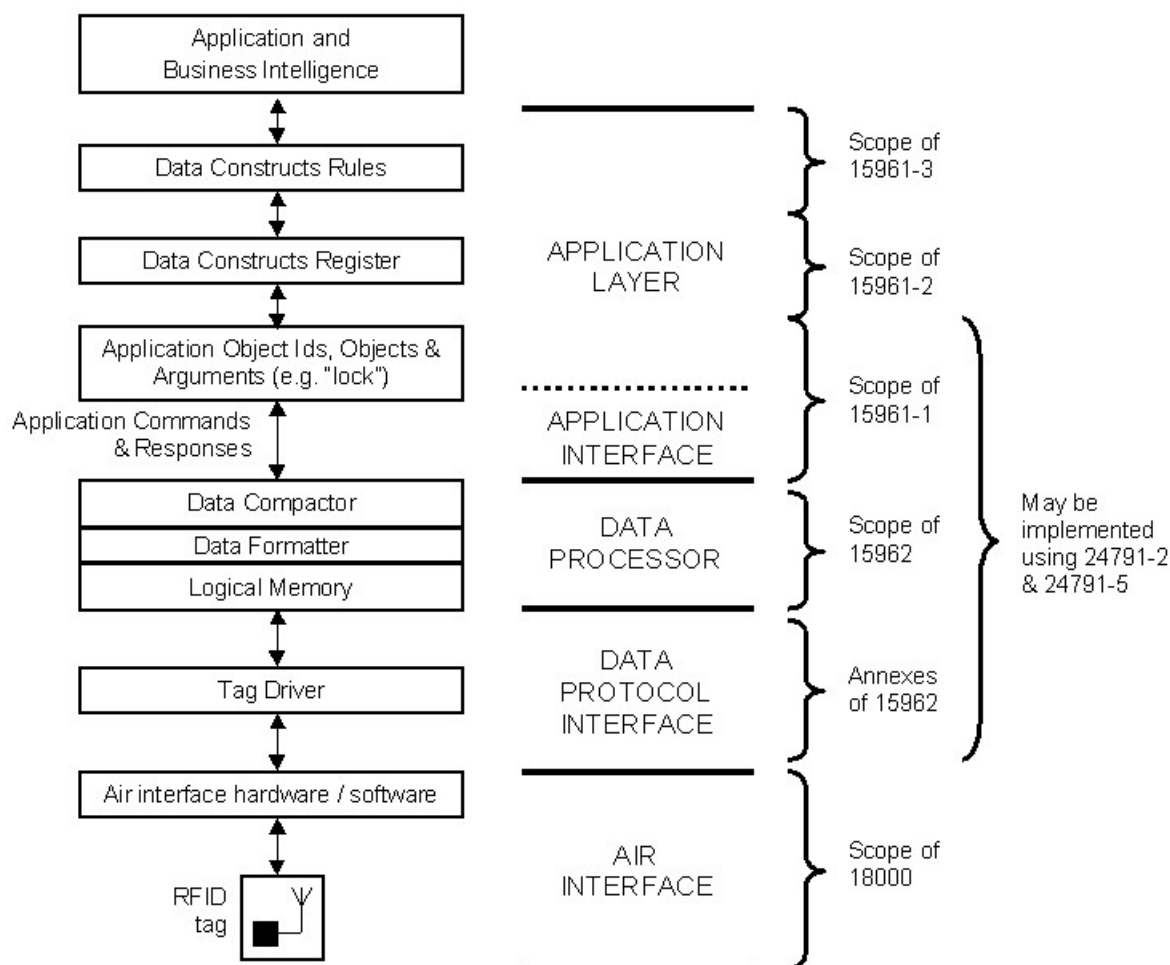


Figure 1 — Schematic of protocol layers for an implementation of RFID for item management

The data protocol specified in ISO/IEC 15961-1, ISO/IEC 15961-2, ISO/IEC 15961-3, and ISO/IEC 15962 is independent of the different RFID tag technologies specified in ISO/IEC 18000, which is concerned with different air interface protocols that function between the interrogator and the RFID tag. This independence is achieved by implementing the standards at different levels in the protocol hierarchy. The RFID data protocol defined in this part of ISO/IEC 15961 is primarily concerned with the upper layers as described below:

5.2.1 Application layer - as defined in the various parts of ISO/IEC 15961

The RFID data protocol specifies how data is presented as objects, each uniquely identified with an Object Identifier, which are meaningful to the application and can be encoded on the RFID tag. ISO/IEC 15961-3 specifies the data construct rules for the AFI, DSFID, Object Identifier for the Unique Item Identifier, and Object Identifier structure for other item-related data. This ensures that each piece of data can be uniquely identified, both within the scope of a particular application and between applications.

Each application needs to be registered according to the rules of ISO/IEC 15961-2 so that the data constructs can be declared and used in an unambiguous manner.

The RFID data protocol in this part of ISO/IEC 15961 defines functions and arguments used to construct application commands and responses. This is so that application programs can specify what data to transfer to and from the RFID tag and to append, update, selectively lock, delete data, or perform other functions on the RFID tag.

To illustrate how the functions and arguments are assembled into a structured format, a number of commands and responses have been constructed using an abstract syntax. This is independent of the host application,

operating system, and program language and also independent of the specific command structures between the interrogator and tag driver. The abstract syntax used in this part of ISO/IEC 15961 is similar to that used in ISO/IEC 24791-5, and is intended to enable closer integration with that standard. The original version of ISO/IEC 15961:2004 included commands defined using ASN.1 abstract syntax. For backward compatibility the commands that were originally defined in this manner have been included in an annex in this part of ISO/IEC 15961.

This RFID data protocol also defines arguments and codes to support responses of data that is read from an RFID tag, including error messages, which are returned to the application.

The abstract syntax may be used as a basis to prepare commands in different programme languages, supporting the functionality and arguments of the abstract commands.

5.2.2 Application interface - as defined in ISO/IEC 15961-1

The application interface may be implemented in a number of different ways that are not explicitly defined in this part of ISO/IEC 15961, nor in ISO/IEC 15962. The basic requirement is to identify data objects distinctly from all others using Object Identifiers, even to enable different data formats to be intermixed on the same RFID tag. The application interface also needs to define command and response arguments unambiguously, so that they can be intermixed with data on the same wired or wireless network.

One major class of implementation, described as a *straight-through process*, is appropriate where the functions and arguments used to construct commands and the arguments and codes used to construct responses, as specified in this part of ISO/IEC 15961, are directly input to the encoding processes of ISO/IEC 15962. Such input can be from computer screens or forms, or more direct transfers from host systems. The advantage of this process is that it avoids the creation of the transfer encoding (see below), but requires more rigorous adherence to the functional requirements of the commands and responses. This part of ISO/IEC 15961 imposes no constraints on the particular application interface process to be adopted, other than the requirement that it be integrated with the encoding rules of ISO/IEC 15962.

An alternative process, consistent with the first edition of ISO/IEC 15961, is to use the abstract syntax for defining the commands and responses in a structured, consistent and verifiable manner, and to generate the transfer encoding that defines the byte stream transferred between the processes of this International Standard and those of ISO/IEC 15962.

Whichever approach is used, the encoding rules of ISO/IEC 15962 shall be followed, and the encoding on the RFID tag has to be compliant with all the arguments in the commands specified in this part of ISO/IEC 15961.

5.2.3 Data Protocol Processing - as defined in ISO/IEC 15962

The RFID data protocol specifies how data is encoded, compacted and formatted on the RFID tag and how this data is retrieved from the RFID tag to be meaningful to the application.

This RFID data protocol provides for a set of schemes that compact the data to make more efficient use of the memory space.

This RFID data protocol also supports various storage formats to enable efficient use of memory and efficient access procedures.

5.2.4 Data Protocol Interface - as defined in ISO/IEC 15962

Each air interface protocol standard in ISO/IEC 18000 has its own specific rules for defining commands and responses. Furthermore, one air interface protocol can support different tag architectures with different memory sizes, and possibly support optional commands. The data protocol provides a mechanism to interface with these rules through specific tag drivers. These allow the basic application commands and responses of this part of ISO/IEC 15961 to be applied independently of the air interface protocol and specific tag architecture.

The tag driver component of the data protocol provides the mapping rule from the generic processes to the specific tag requirements. These mapping rules are used to write data and to read data.

Additional tag drivers can be specified as new air interface protocols are introduced in the ISO/IEC 18000 series of standards.

5.3 Flexible implementation configurations

This RFID data protocol specifies the application level communication and the RFID tag interrogator level rules for data encoding, compaction and storage formats. This protocol may be implemented:

- with ISO/IEC 15962 incorporated into the software system infrastructure architecture defined in ISO/IEC 24791. This is the recommended approach for any networked application.
- with this part of ISO/IEC 15961 and ISO/IEC 15962 incorporated into stand-alone software or devices that have as its output conformant encoding and / or conformant decoding with responses compliant to the responses of this part of ISO/IEC 15961.

5.4 Functional processes – interrogator implementation

There are various functional processes that need to take place to write data to an RFID tag and to read data from it. Figure 2 — Logical functions and interfaces of ISO/IEC 15962 with other RFID system components shows a schematic of an example implementation where the processing of the data protocol resides in the interrogator. This illustration is provided to help with the understanding of the processes, and although a typical implementation, many others are possibly compliant with this data protocol.

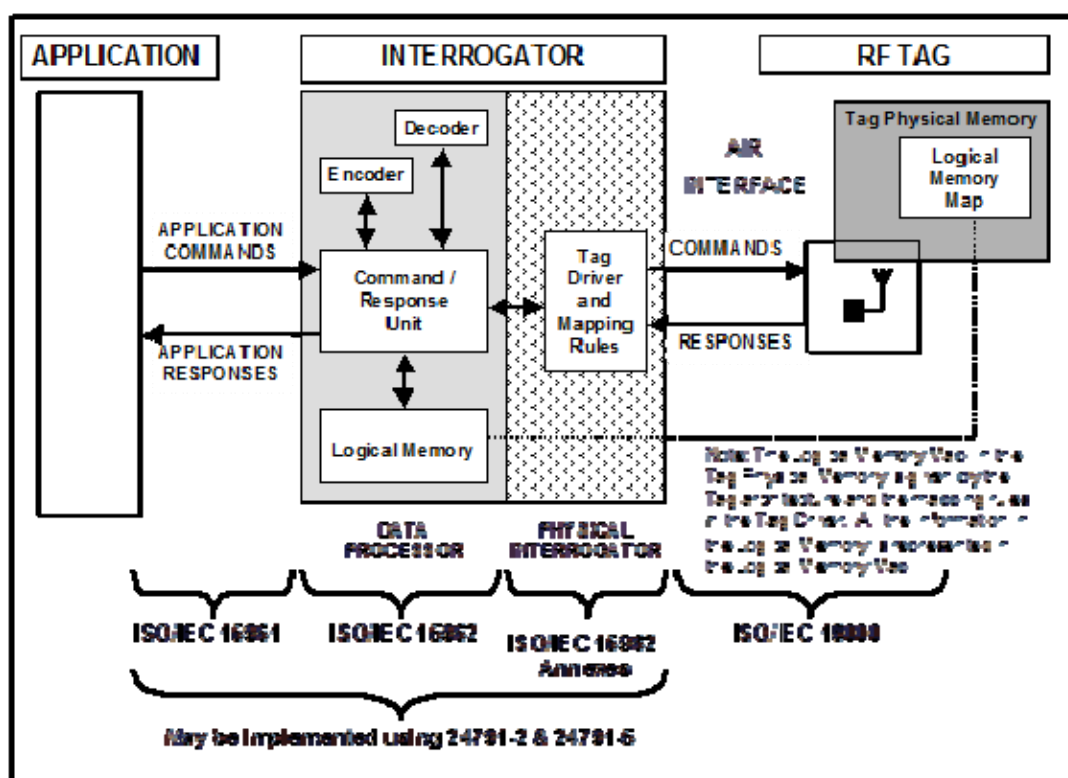


Figure 2 — Logical functions and interfaces of ISO/IEC 15962 with other RFID system components

5.4.1 Functional processes - application interface

The data flows between the application and the Data Processor are formatted according to this part of ISO/IEC 15961 and are uncompact. However, there are numerous established systems where data is formatted to be compliant, for example, with a bar code related syntax. It is therefore reasonable to insert interface modules in the data flow to convert from and to existing application formats.

NOTE Careful consideration should be given to the extent that established systems need to be supported relative to the potential benefits to be gained from adopting the data protocol specified in this part of ISO/IEC 15961 and ISO/IEC 15962. This is because this protocol has been developed around the features of RFID, such as selective read/write and the ability to lock data. Older protocols are unlikely to support such features.

5.4.2 Functional processes – interrogator

In the process illustrated in Figure 2, the interrogator is the module in which all the basic processing of the data protocol takes place and there is an interface to the RFID tag. Different implementations might separate some of the functions described below and have an interface between the application and the physical interrogator.

5.4.2.1 Data Processor

The Data Processor provides all the processing, which is as specified in ISO/IEC 15962 and is required for handling application data. It consists of the following components, all of which are described more fully below: Command/Response Unit, Logical Memory, Encoder (which supports a Data Compactor and Formatter function) and Decoder (which supports the inverse functions of the Encoder). The Data Processor can physically reside anywhere between the application software and the tag driver but shall contain all the components. Some, or all, of the Data Processor functions may be implemented using processes defined in ISO/IEC 24791-2 and ISO/IEC 24791-5.

5.4.2.1.1 Command/Response unit

The Command/Response Unit receives the application commands from the application in a format specified in this part of ISO/IEC 15961, acting upon these commands where appropriate and converting to the specific RFID tag lower level command codes.

EXAMPLE

An application command of *write Data Object {name}* is application related. The data protocol recognises this and can format the data onto the Logical Memory in the Data Processor. Information from the particular RFID tag is required to set the parameters of the Logical Memory Map (e.g. number of bytes, whether a directory is in use, etc) on the RFID Tag. The Tag Driver converts the application command into a tag-specific command.

It can be seen from this example that there is a distinct logical boundary between the Data Processor and the Tag Driver.

5.4.2.1.2 Logical Memory

This is an array of contiguous bytes of memory acting as a common software representation of the Logical Memory Map in the user memory of the RFID tag to which the Object Identifiers and data Objects are mapped in bytes. The Logical Memory takes into account some parameters of the real RFID tag, for example the block size, the number of blocks and the storage format. The Logical Memory ignores any detailed tag architecture.

The use of the Logical Memory means that an application can interface with an application-compliant RFID tag, but that individual RFID tags can have completely different memory capacities and architectures. This enables an implementation to benefit from new technological developments permitted within the framework of ISO/IEC 18000, such as larger capacity or faster access RFID tags, without changing the application.

5.4.2.1.3 Encoder

The Encoder controls the process of writing data through the functional processes performed by the Data Compactor Module and Formatter Module.

5.4.2.1.4 Data Compactor

The Data Compactor provides the standard compaction rules to reduce the number of bytes stored on the RFID tag and transferred across the air interface. Numeric data, for example, is octet based to some coded character set for the application, but can be encoded in a compact form on the RFID tag memory.

5.4.2.1.5 Formatter

The Formatter provides the processes to place the Object Identifier and Object (data) into an appropriate and efficient format for storing on the Logical Memory.

NOTE The physical mapping of bits to comply with the RFID tag architecture is performed based on information provided by the Tag Driver.

5.4.2.1.6 Decoder

The Decoder controls the process of reading and interpreting data through the functional processes performed by the Data De-compactor Module and De-formatter Module.

5.4.2.2 Tag Driver

The Tag Driver provides two main functions:

- It provides mapping rules on the structure of the RFID tag to enable the contents of the Logical Memory in the Data Processor to be exchanged with the Logical Memory Map of the RFID tag in use.
- It provides facilities that accept the application commands of this data protocol, and converts them to a format that results in calls to command codes supported by the particular RFID Tag. For example, an application command **write Data Object {name}** could result in the RFID tag related command of write (block #, data).

The description of the tag driver for particular RFID tags is provided in annexes of ISO/IEC 15962. For the purpose of ISO/IEC 15962, a tag driver is unique to a particular air interface type of RFID tag as specified in the appropriate part of ISO/IEC 18000. This is a logical representation; physical implementation could combine features of different logical tag drivers. An interrogator may support one or many tag drivers.

5.4.2.3 Transfer mechanisms

The transfer mechanisms for transferring data, commands and responses between the Data Processor (i.e. the implementation of this part of ISO/IEC 15961) and the RFID tag need to be done through the interrogator or incorporated within the functions of the interrogator.

5.4.3 RFID tag

Although the RFID tag is beyond the scope of this International Standard, the tag is shown in Figure 2 — Logical functions and interfaces of ISO/IEC 15962 with other RFID system components to complete the flow of data and commands. Within the RFID tag, the Logical Memory Map represents all the data in the Logical Memory of the Data Processor converted (or mapped) to a location structure determined by the mapping rules in the Tag Driver and the architecture of the RFID tag.

5.5 ISO/IEC 15962 and the Data Processor

ISO/IEC 15962 defines all the rules for encoding data on an RFID tag. The implementation of these rules is described in this part of ISO/IEC 15961 as the Data Processor. As described in 5.4, various processes are undertaken to achieve successful encoding. The rules ensure the encoded bytes are 'self-declaring' in a reading operation without any previous or independent knowledge of what is encoded on the tag. This allows the Data Protocol to be used in open systems where the organisation encoding the data on the RFID tag might be completely unaware of what organisation might eventually read the data from the RFID tag. This is achieved by:

- Having clearly defined rules for compacting data that are applied independently of the application and the type of RFID tag.
- Defining fundamental structuring rules for the encoding of data largely determined by a user-selectable feature called the **Access-Method**.
- Having precise structuring rules within each **Access-Method**, including a clearly defined syntax that allows reading systems to selectively identify data required for the application.

Clause 7.2.4 provides a description and a definition sufficient for an application administrator to evaluate the merits of the four **Access-Methods** currently supported. The associated processing is fully specified in ISO/IEC 15962.

6 Presentation conventions

6.1 Presentation of commands, responses and arguments

6.1.1 Commands and responses

Commands and responses are defined within a box format with a double line border, such as is shown in Figure 3 — Box format for commands and responses.



Figure 3 — Box format for commands and responses

Each command, or response, contains an ordered list of fields or arguments. The field/argument names are shown in **bold** type face using this format: **Argument-Name**. When necessary, the field/argument name is followed by the data type and a brief description. Fields with values that are restricted to a subset of the range of their data types have their possible and legal values shown in *italics* below the field name.

NOTE None of this applies to the original commands and responses when presented in the ASN.1 abstract syntax as in ISO/IEC 15961:2004.

6.1.2 Arguments

The more complex arguments, containing sub-arguments, are defined within a box format with a single line border, such as is shown in Figure 4 — Box format for arguments.



Figure 4 — Box format for arguments

6.1.3 Data types

The following data types are used in the commands and responses:

- BOOLEAN: An argument that can have the values TRUE or FALSE.
- BIT STRING: A sequence of bits.
- BYTE: An integer with the possible values 0 to 255, usually expressed as a hexadecimal value 00_{16} to FF_{16} .
- BYTE STRING: A sequence of bytes. (Equivalent to OCTET STRING)
- EBV-8: A binary method to encode variable size numbers in the same field by using a leading indicator bit preceding a 7-bit value. The final EBV-8 component begins with a 0, all preceding components begin with a 1. For example, $69_{10} = 01000101$, whereas $369_{10} = 101110001_2$ whose EBV-8 representation is **10000010 01110001** (i.e. separated into 7-bit stings and then the indicator added as a prefix. The only requirement for using an EBV-8 code in a command is where this type of value is returned in an air interface response.
- HEXADECEMAL ADDRESS: A location (on the memory of an RFID tag), expressed as a hexadecimal value
- INTEGER: An integer can take any whole number. In the context of this part of ISO/IEC 15961, the values are all positive.
- OBJECT IDENTIFIER: An Object Identifier as defined in 6.2.1.

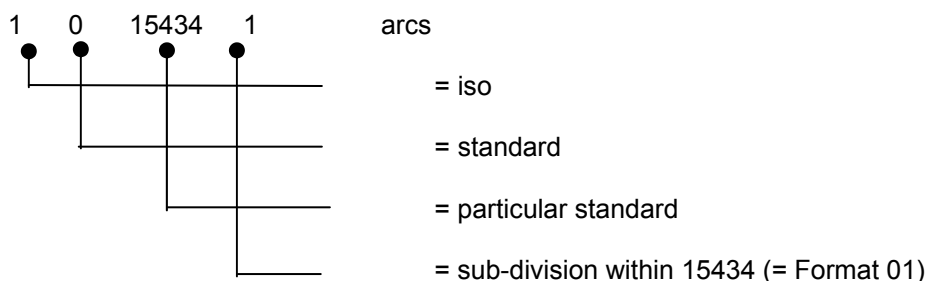
6.2 Object Identifier presentation in the application interface

6.2.1 Object identifier structure to ISO/IEC 8824-1

This part of ISO/IEC 15961 uses the OBJECT IDENTIFIER type as defined in ISO/IEC 8824-1 and with identifiers assigned as specified in ISO/IEC 9834-1. This uses a registration tree with a common implied root node (ISO/IEC 9834-1), a series of arcs from each node, with new arcs added as required to define a particular Object. Thus, the body responsible for a particular node:

- has a defined set of arcs to identify itself
- can manage the allocation of arcs under its node, independently of other bodies
- is assured of uniqueness from all other arcs in the registration tree

EXAMPLE



The only top arcs permitted for all Object Identifiers are shown in Table 1 — Object Identifier Top Arcs.

Table 1 — Object Identifier Top Arcs

Identifier Arc Name	Numeric Value
itu-t	0
iso	1
joint-iso-itu-t	2

NOTE Any ISO/IEC standard, such as this part of ISO/IEC 15961, have Object Identifiers under the ISO top arc.

The second arc is administered by the relevant organisation named for the top arc. The current list of top and second arcs is given in Part 3 of ISO/IEC 15961.

The third arc is controlled by the system or body defined for the second arc; sometimes this is a Registration Authority. The hierarchical structure continues until the Object is identified uniquely. The procedure of naming Object Identifiers ensures that each Object is unique within its "parent" arc and that each parent arc is unique within its previous level, right back to the top 3 arcs.

NOTE This structure enables Object Identifiers from different domains (e.g. open and closed systems) to be encoded unambiguously on an RFID tag memory.

Three forms of Object Identifier are used with the RFID Data Protocol:

- **Object-Identifier:** This full structure is used for communications between the application and the Data Processor defined by the scope of this part of ISO/IEC 15961. The full structure is suitable for use in ISO/IEC 15962 where the set of Object Identifiers to be encoded on the RFID tag have different higher level arcs, or where a generic system is in place.
- **Relative-OID:** This structure is used in conjunction with the **Root-OID** (see below) for communication between the application and the Data Processor defined by the scope of this part of ISO/IEC 15961. These structures are applied in situations where a common root applies to the set of Object Identifiers to be encoded on the RFID tag. For example, if all the Object Identifiers have the common root 1 0 15961 12 encoding space can be saved on the RFID tag if this common **Root-OID** does not have to be encoded for each Object Identifier. The **Relative-OID** is a suffix to a common **Root-OID**, which is either encoded or declared in some other way.

NOTE Except in the command and response abstract forms (see Clause 10) and places where the distinction is vital, the term **Object-Identifier** applies also to the **Relative-OID**.

- **Root-OID:** The **Root-OID** is the common part of a set of encoded Object Identifiers. It acts as a common prefix to the **Relative-OID** values encoded on the RFID tag. This structure is particularly important in applications that require a variety of data from a common data dictionary to be encoded on an RFID tag. The **Root-OID** is either explicitly encoded or declared in some other way, according to the encoding rules of ISO/IEC 15962.

6.2.2 Presenting the Object-Identifier in the style of ISO/IEC 8824-1

When the **Object-Identifier** is presented in the style of ISO/IEC 8824-1, spaces are inserted between each arc as follows:

1 0 15961 12 1

NOTE ISO/IEC 8824-1:2003 clause 31.12 shows a set of equivalent notations including the way as shown above. The more formal representation of this in ASN.1 is: {iso(1) standard(0) rfid-data-protocol(15961) iata(12) baggage-id (1)}

6.2.3 Presenting the Object-Identifier as a Uniform Resource Name (URN)

The **Object-Identifier** may also be presented in the URN in the following format, based on IETF RFC 3061, with the decimal point character between each arc:

urn:oid:1.0.15961.12.1

6.3 Byte Notation

6.3.1 The byte: the basic unit for 8-bit coding

This part of ISO/IEC 15961 supports binary, 7-bit, 8-bit and user data that may exceed 8-bits per character. The common unit of coding is the 8-bit byte (also known as the octet). Binary data shall be padded with leading zero bits until the binary value is octet aligned; 7-bit data shall be represented as bytes with bit 8 (see 6.3.2) set to a zero value. Data exceeding 8-bits shall be encoded in multiple bytes.

A byte is represented as 2 hexadecimal characters with the values 0-9, A-F.

6.3.2 Bit ordering

Within each byte, the most significant bit is bit 8 and the least significant is bit 1. Accordingly, the weight allocated to each bit is as shown in Table 2 — Bit sequence in the 8-bit byte.

Table 2 — Bit sequence in the 8-bit byte

Bit Value	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1
Weight	128	64	32	16	8	4	2	1

6.3.3 Byte conversion

The 8-bit value is converted into the two hexadecimal characters with bit 8, bit 7, bit 6 and bit 5 having the weights 8, 4, 2 and 1 respectively to define the first hexadecimal character. Bit 4, bit 3, bit 2 and bit 1 retain the weights 8, 4, 2 and 1 respectively to define the second hexadecimal character.

7 Processing application commands and responses

7.1 General

Since the publication of the first edition of ISO/IEC 15961 and ISO/IEC 15962, it has become clear that a number of device configurations do not require the transfer encoding to properly support the functions of the Data Protocol. These device configurations have similar characteristics in that they can support all processes internally (examples include printer-encoders, hand-held RFID readers, and software packages), but other device configurations can have similar characteristics. These characteristics include a complete encoding process that supports the functionality of the application command and the encoding processes on the logical memory and/or the decoding processes from the logical memory to the application responses.

In these devices, transfer encoding would have to be created, and immediately be decoded and discarded, resulting in two unnecessary processes that add to the process time and complexity. As part of the revision of ISO/IEC 15961 into a multi-part standard, this part of ISO/IEC 15961 now includes support for a "straight through" process.

NOTE The distinct choice between options that support transfer encoding and a straight through process is now more clearly defined than in the first edition of ISO/IEC 15961.

There are two fundamentally different ways to implement support for the commands and responses defined in Clause 10.

OPTION A: Straight through process

In devices or software that incorporate data input of commands compatible with this part of ISO/IEC 15961 and also support the encoding processes of ISO/IEC 15962, the transfer encoding is not required. It is therefore essential to ensure that the basic functionality of the commands and responses are followed. The rules defined for Object Identifiers (see 7.3.2) and the command and response arguments (see 7.4) shall be followed.

Various forms of implementation are possible for this straight through processing of the Data Protocol, including input from forms and more direct transfer from host systems. This part of ISO/IEC 15961 places no constraints on the process adopted.

OPTION B: Using transfer encoding

If a device, or software, only supports the processes of this part of ISO/IEC 15961 or only supports the encoding processes of ISO/IEC 15962, then the original transfer encoding may be used (see Annex A). Alternative transfer encoding including the binary encoding defined in ISO/IEC 24791-5 may be used for better integration with systems that support such interface mechanisms. In turn, this means that the command and response structures defined in Clause 7 shall also be considered as abstract or functional definitions of the requirements to properly encoded data on an RFID tag, or read and interpret data from the tag.

7.2 Encoding system related information in commands

7.2.1 Singulation-Id

The **Singulation-Id** (previously defined as TagId in ISO/IEC 15961:2004) is provided by the Tag Driver, and identifies the RFID tag unambiguously for at least the period of a data transaction.

In the data protocol, the **Singulation-Id** shall be up to 255 bytes long and acts as a file reference for the Logical Memory, and in turn provides a one-to-one link to the Logical Memory Map of the RFID tag itself. **Singulation-Id** shall be based upon one of the following, the choice being determined by the design of the Tag Driver and the air interface protocol:

- a) A completely Unique Item Identifier programmed in the RFID tag, as specified in the ISO/IEC 18000 series. In that series of standards it is defined as the Tag ID.
- b) A data related identifier (e.g. like a Unique Item Identifier (UII)), that provides for uniqueness within the particular application domain of item management. This requires the UII to be read to establish the **Singulation-Id**.
- c) A virtual or session identifier based on a time slot or other feature managed by the air interface protocol.
- d) Combinations of (b) and (c), e.g. a virtual **Singulation-Id** across the air interface, but requiring the data related identifier to be returned as a response.

7.2.2 AFI

The **AFI** (previously defined as ApplicationFamilyId in ISO/IEC 15961:2004) refers to specific identifiers that enable selective addressing of RFID tags. This may be supported by a mechanism at the air interface. Because the **AFI** is supported by standards for smart card, the structure that follows is non-conflicting with those standards. The value of the **AFI** for RFID for Item Management shall be a single byte value specified in Part 3 of ISO/IEC 15961, as follows:

- If in the range 00_{16} to $0F_{16}$ the code applies to a closed system application, as defined in Part 3 of ISO/IEC 15961. The list of code values is updated in the associated RFID data constructs register.
- If in the range 90_{16} to CE_{16} the code applies to an open system application as defined in Part 3 of ISO/IEC 15961. The list of code values is updated in the associated RFID data constructs register.
- The value CF_{16} is reserved as an extension code for multiple byte **AFI** code values.

The **AFI** should be stored on the RFID tag in some form, or alternatively may be determined by the air interface services if these are sufficiently specific. If the **AFI** is not supported by a class of RFID tag, and the services from the air interface do not provide this by other means, the value of this code in the system information for the RFID tag shall be 00_{16} .

NOTE In the first edition of ISO/IEC 15961 (and in the original commands and responses) the value of ApplicationFamilyId consists of two integer values concatenated into a single octet when stored within system information on the RFID tag. The first integer refers to the applicationFamily and the following codes shall apply:

- | | |
|---------|---------------------------------------|
| 0 | for closed systems |
| 9 to 12 | as defined in Part 3 of ISO/IEC 15961 |

The second integer of the ApplicationFamilyId, for the applicationFamilies 0 and 9 to 12, refers to the applicationSubFamily. The integer values convert to a value equivalent to the single byte **AFI**.

7.2.3 DSFID

The **DSFID** (previously defined as StorageFormat in ISO/IEC 15961:2004) refers to specific identifiers that enable efficient encoding on RFID tags. The value of the **DSFID** for RFID for Item Management is a single byte, or multiple byte. It encodes the **Access-Method** and **Data-Format**, both of which are defined below. The **Access-Method** is encoded in bits 8 and 7 of the **DSFID**, with the final five bits (bits 5 to 1) encoding the **Data-Format**. Bit 6 defines additional functionality supported by some types of RFID tag.

If the **Access-Method** has a value between 4 and 15, extensions to the **DSFID** are implemented by the Data Processor.

NOTE None of these values has yet to be assigned, and such assignments will only be by amendment to this part of ISO/IEC 15961.

If the **Data-Format** has a value between 32 and 287 as defined in Part 3 of ISO/IEC 15961, extensions to the **DSFID** are implemented by the Data Processor. As the **Data-Format** is registered as part of the ISO/IEC 15961-2 Data Constructs register, this extension occurs independently of changes to this part of ISO/IEC 15961.

7.2.4 Access-Method

The **Access-Method** defines the manner in which data can be mapped on the RFID tag and be accessed from the RFID tag. The value of **Access-Method** should be stored on the RFID tag, or may be defined by the air interface services, if this can be done unambiguously. The **Access-Method** is defined as a bit value and the following codes shall apply:

- 0 **No-Directory** - This **Access-Method** provides the simplest set of encoding rules. The basic rule is to link a data **Object** to an **Object-Identifier**, and support this with appropriate syntactical components to create a Data-Set. The **Data-Sets** are concatenated to produce a contiguous sequence of encoded bytes in the Logical Memory Map.
- 1 **Directory** - The **Directory** structure supports all the encoding rules of the **No-Directory** structure, but additionally supports the encoding of a directory at the higher address values within the Logical Memory Map on the RFID tag. When the **Directory** structure is used, any command seeking to selectively read data can first access the directory to find the memory address of the beginning of the **Data-Set**, and then go to that location to read the bytes that represent the encoding of the **Data-Set**.

NOTE The Directory may be written to the RFID at a later time than the initial encoded data.

- 2 Packed-Objects** - The **Packed-Object** structure has been introduced at the first revision of ISO/IEC 15962 and this edition of this part of ISO/IEC 15961. The encoding scheme is fundamentally different, because it takes a set of **Object-Identifiers** and their **Objects** and encodes them in an indexed structure that integrates compaction and encoding. The **Packed-Object** encoding scheme requires a rule-based table for each **Data-Format**, which calls up specific compaction schemes for the individual elements. The compaction schemes are standard, irrespective of the **Data-Format**. The **Packed-Object** scheme requires more complex encoding and decoding rules than the **No-Directory** and **Directory Access-Methods**, but offers significant encoding efficiency even over the basic **No-Directory** structure when a number of **Object-Identifiers** need to be encoded. This can reduce the amount of memory required on the RFID tag, and can reduce the size of message transferred across the air interface in response to a read command. Application domains need to define a table of indexed **Relative-OIDs** for this **Access-Method** to be implemented. It can only be applied as an alternative to the **No-Directory** or **Directory Access-Methods** on a particular RFID tag; but tags with any of the **Access-Methods** may be intermixed in the same application.
- 3 Tag-Data-Profile** - The **Tag-Data-Profile** scheme is intended to support fixed message structures, typically where a number of **Object-Identifiers** and their **Objects** need to be encoded. The fixed message structure is best suited for applications that are reasonably homogeneous and additionally have a consistent requirement for exactly the same **Object-Identifiers** to be encoded on all RFID tags in the application. Each **Tag-Data-Profile** requires registration.
- 4 Multiple-Records** - The **Multiple-Records** encoding process overlays a structure onto the **No-Directory**, **Packed-Objects**, and **Tag-Data-Profile Access-Methods** for multiple instances of these, and even a mixture of these **Access-Methods**, to be encoded in the same Logical Memory. This is achieved through the introduction of an encoded MR-header and a preamble for each record. This allows the encoding of individual records to fully comply with the inherent **Access-Method**. This also enables the same **Object-Identifier** to be encoded in separate records, with one rule that supports a list of the same data element in one record. There are three main classes of application supported by the **Access-Method** and these can also be intermixed. One class enables different data formats and owners of data to share the same RFID tag, but under the control of the original owner of the tag. Another supports a time sequence of history records of the same type to be repeated. The third major class is for the support of hierarchically related records e.g. for supply chain delivery or bill of material types of structure.
- 5 to 15** Reserved for future encoding schemes to be defined in ISO/IEC 15962 and invoked under rules for a multiple byte **DSFID**. Some of the new **Access-Methods** might be possible to be inherent encoding schemes in **Multiple-Records**. This can only be determined as the encoding schemes are developed.

Where a choice can be made, the following guidelines can be of assistance:

- The **No-Directory** structure is better suited to RFID tags with small memory capacity, because the directory itself is an overhead that needs to be encoded. It also is better suited where there are few **Object-Identifiers** to be encoded, so that a continuous read function will transfer all the encoding (or at least sufficient of the encoded bytes) to enable the bytes to be parsed to find the required **Object-Identifier** and associated **Object**.
- The **Directory** structure is obviously better suited where the conditions differ from those suited to the **No-Directory** structure. In addition, it is better suited to applications that call for selective reading, writing, or modifying one or few **Object-Identifiers** from among many. In this type of situation, the extra read processes to transfer the directory to the Data Processor are likely to be balanced by the shorter read time for the selected **Object-Identifier**.
- The **Packed-Objects** structure requires more complex encoding and decoding, but offers a significant improvement in encoding efficiency over the basic **No-Directory** structure. As such, it is better suited to applications where the encoding requirements are relatively high compared to the memory capacity of the RFID tags used in the application.

- The **Tag-Data-Profile** scheme is best suited to applications where the domain is homogeneous (e.g. a vertically integrated supply chain where all suppliers are known to all customers) and all parties agree that each **Object-Identifier** shall be encoded and agree to a common structure and format for each data **Object**.

As the **Access-Method** has to be specified by the application, it should be possible to measure transfers of data that simulate a directory plus data structure with that of a **No-Directory** structure. Such a test can be used to determine a reasonable breakeven point between this and the other **Access-Methods**, taking into account the variety and length of data and the typical read/write implementations expected for an application.

Once an **Access-Method** has been specified for the RFID tag, the application does not need to qualify how reading, writing, or the organisation of the bytes on the Logical Memory is to be achieved. This is the function of the Data Processor and the Tag Driver.

7.2.5 Data-Format

The **Data-Format** defines the **Root-OID** of the **Object-Identifiers** being stored on the RFID tag. The **Data-Format** enables data Objects to be identifiers simply by a **Relative-OID**, making more efficient use of encoding space or to restrict data to one class.

Full **Object-Identifiers** (i.e. with a different **Root-OID** to that define by the **Data-Format**) may still be encoded on the RFID tag, enabling data from different application domains to be encoded, albeit without the same level of encoding efficiency.

The value of **Data-Format** should be stored on the RFID tag or may be defined by the air interface services if this can be done unambiguously. When defined by the air interface protocol, there shall be a one-to-one relationship with the type of RFID tag and the value of **Data-Format**. The value of the **Data-Format** for RFID for Item Management shall be an integer value in the range 0 to 287 specified in Part 3 of ISO/IEC 15961. The following are relevant to this part of ISO/IEC 15961:

- | | |
|---------|---|
| 0 | Not-Formatted - This is used for RFID tags not formatted, or not yet formatted, to this part of ISO/IEC 15961. |
| 1 | Full-Featured - This Data-Format supports any type of data format where the full Object-Identifier is used. Its prime purpose is to enable heterogeneous data (i.e. from different data dictionaries) to be encoded on the one RFID tag. For example it could be used to encode data from different closed system applications unambiguously, using the ISO/IEC 9834-1 registered Object Identifiers for each application. |
| 2 | Root-Oid-Encoded - This Data-Format is used when all the data on the RFID tag has a common Root-OID , which does not comply with one of the specific Root-OIDs associated with a Data-Format registered under the rules of Part 2 of ISO/IEC 15961.

The Root-OID for Data-Format 2 shall be directly encoded on the RFID tag using the appropriate root arcs. Each Object is encoded on the RFID tag using a Relative-OID , representing the remaining lower order arcs. |
| 3 to 28 | These code values applies to an open system application as registered under Part 2 of ISO/IEC 15961. The list of code values is updated in the associated RFID data constructs register. |
| 29 | This code applies to closed system data where the encoded data is compliant to the rules of this part of ISO/IEC 15961 and ISO/IEC 15962. |
| 30 | This code applies to closed system data, where the encoding rules are not compliant with ISO/IEC 15962. It is of use for applications that are migrating from other encoding rules, enabling data to be distinguished and properly encoded. It is also of use prior to the transition of RFID tags from a closed system application to an open system application. |

- 31 This value is not assigned to any application, because its binary equivalent is used to signal an extension to the **Data-Format**, in the range 32 to 287.
- 32 to 287 These values are reserved for open system applications as registered under Part 2 of ISO/IEC 15961, when an extension mechanism is invoked under ISO/IEC 15962 rules for a multiple byte **DSFID**.

7.3 Preparing the basic Objects and other application-based arguments

This is an initial process to ensure that the data **Objects** are prepared in a format compatible with this part of ISO/IEC 15961. It is recognised that there are message-based protocols and syntax for existing AIDC application standards that differ from the Object-based protocol of this part of ISO/IEC 15961. It is possible to achieve the benefits of the Object-based protocol and maintain compatibility with message based systems by converting between the two formats (see Annex B).

The outputs described in the sub-clauses that follow shall be the format of inputs to the Data Processor. However, the mechanisms to achieve them on input and output are not. The requirement is to assign an **Object-Identifier** to each **Object** using the data dictionary relevant to the application standard.

7.3.1 General model

Figure 5 — Data flow model: Prepare basic Objects provides a data flow model for preparing each **Object-Identifier** and associated data **Object**. Each type of application data, and particularly those covered by this part of ISO/IEC 15961, has a data dictionary, or list of data Objects or data elements.

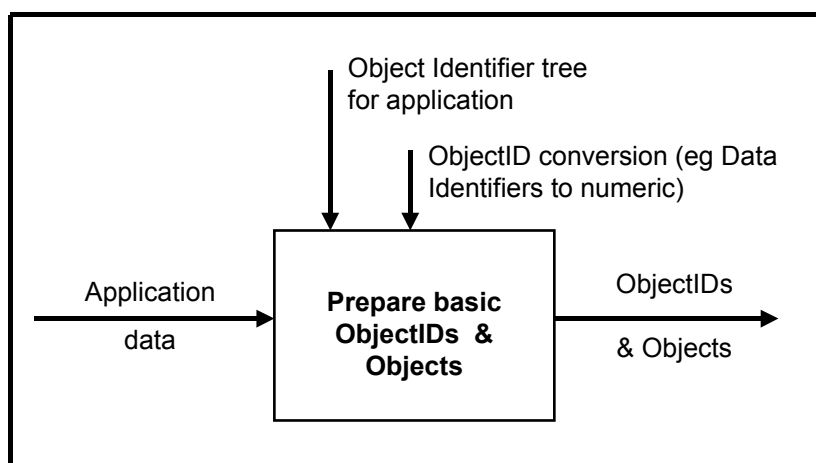


Figure 5 — Data flow model: Prepare basic Objects

Any such data dictionary may evolve over a period of time and maintenance of such a data dictionary shall be independent of this part of ISO/IEC 15961. The data dictionary usually consists of a coded list (numeric, alphabetic, alphanumeric, etc) of data Objects and their specification for use within the domain of the application standard. Part 3 of ISO/IEC 15961 and its associated RFID data constructs register defines **Object-Identifiers** that are relevant to particular applications.

7.3.2 Object-Identifier

The **Object-Identifier** shall be provided by the application system as a series of arcs (as defined in 6.2), which comply with ISO/IEC 9834-1, including all the **Object-Identifiers** associated with the established **Data-Formats**. The **Object-Identifier** should be presented as a (full) **Object-Identifier** in the application commands, unless an application-specific system supports all the relevant **Relative-OIDs**. Whenever possible, the Data Processor converts a (full) **Object-Identifier** to a **Relative-OID** for more efficient encoding on the RFID tag, based on the **Data-Format** received from the RFID tag.

7.3.3 Relating Object-Identifiers

Message based syntax can use recursive or looping techniques to create repeated sequences of related data (e.g. individual quantity and batch numbers linked to different product codes). When the complete message is parsed, the syntax identifies boundary points so that the attributes are correctly linked to the primary code.

With an Object-based system (such as the Data Protocol of this part of ISO/IEC 15961 and ISO/IEC 15962) operating at a base level, there is a risk of creating false links (i.e. product code A could be linked to quantity of product B). The problem can be overcome using one of the techniques described in Annex C.

7.3.4 Object

The application provides data in the form of an **Object**. This value is byte based, and associated with the definition of the **Object-Identifier** provided by the application data dictionary. Specific advice about particular interpretation is given in the following sub-clauses.

7.3.4.1 Basic 8-bit character set information

Many business applications make use of ISO/IEC 8859-1 or its subset ISO/IEC 646. As such, the interpretation (or presentation) of the data is known to all participants in the system. This includes the specific sender and recipient even, if as is common in an open AIDC system, they are not known directly to each other. If text file and browser application software are set up to handle the ISO/IEC 8859-1 character set, most of the data content can be displayed as intended. If the content has a specific interpretation, the RFID application needs to establish between the participants exactly how particular data is to be interpreted. This can be achieved by publishing a data dictionary using the **Object-Identifiers** as a reference.

7.3.4.2 Support for ISO/IEC 10646

ISO/IEC 10646 supports the character glyphs of all character sets. One option to support data in any other specific character set is to pre-process this by converting to ISO/IEC 10646 using mapping tools, and then compacting this to UTF-8 encoding (as defined in ISO/IEC 10646). Any data directly defined by ISO/IEC 10646 should also be UTF-8 encoded, to reduce the number of bytes required to store the **Object** in the Logical Memory Map.

7.3.4.3 Support for secure or encrypted data

All forms of secure data, including encrypted data, shall be created by the application prior to being passed as an **Object** to the Data Processor. This is done for two reasons:

- It requires security of data to reside with the application and to be changed independently of the Data Processor.
- It allows the Data Processor to handle all data, whether encrypted or not, in a similar manner.

The fact that data is encrypted may need to be made known to the entire set of systems users, but the actual method of encryption can be restricted to the sender and intended recipient.

Further details are provided in 7.6.

7.3.5 Compact-Parameter

The inclusion of **Compact-Parameter** in a command shall determine whether the application data is to be compacted by the Data Processor. The **Compact-Parameter** is an integer value and the following codes apply.

- 0 **Application-Defined** - The **Object** shall not be processed through the data compaction rules of ISO/IEC 15962 and remains unaltered when stored in the Logical Memory Map of the RFID tag.
- 1 **Compact** - This requires using the basic ISO/IEC 15962 compaction rules to compact the **Object** as efficiently as possible to reduce the number of bytes required on the Logical Memory Map.
- 2 **UTF8-Data** - This identifies that the **Object** has been externally transformed from the ISO/IEC 10646 coded character set to UTF-8 encoding. The **Object** shall not be processed through the data compaction rules of ISO/IEC 15962 and remains unaltered for transfer to the Logical Memory Map.
- 3 **Pack-Objects** - This identifies that a set of Objects is to be encoded using the Packed Object encoding scheme and identified with the associated **Access-Method**. The set of Objects shall not be compacted using the basic ISO/IEC 15962 compaction rules, but use the Packed Object encoding rules.
- 4 **Tag-Data-Profile** - This identifies that a set of **Objects** is to be encoded using the Tag Data Profile encoding scheme and identified with the associated **Access-Method**. Although the set of compaction schemes is identical to the basic ISO/IEC 15962 compaction rules, the Profile IDTable specifies a particular compaction scheme. The Profile IDTable specification shall be followed.
- 5 **Monomorphic-UII** - This identifies that a single **Object** defining a UII is to be encoded using the compaction rules defined by the **AFI**. As this is applied to a UII, this **Compact-Parameter** shall only be applied to a single **Object** per memory bank.

NOTE This **Compact-Parameter** is explicitly defined because for some tags the encoding of **Objects** is independent of the encoding of the **AFI**, so the Data Processor shall access information about the **Compact-Parameter** from the Data Constructs register as defined by Part 2 of ISO/IEC 15961.

6 to 13 reserved for future definition

- 14 **De-Compacted-Monomorphic-UII** - This identifies that the **Object** in a response has been de-compacted using the rules defined by the **AFI** assigned for a particular **Monomorphic-UII**.

NOTE This **Compact-Parameter** is explicitly defined because the **Monomorphic-UII** does not have an associated **Object-Identifier** encoded on the tag. The **Object-Identifier**, as defined by the **AFI**, needs to be added to the response. The Data Processor shall access information about the **Object-Identifier** from the Data Constructs register as defined by Part 2 of ISO/IEC 15961.

- 15 **De-Compacted-Data** - This identifies that the **Object** in a response has been de-compacted using rules in ISO/IEC 15962 and restored to its original application input format.

Generally, compaction should be applied because it increases the encodation efficiency on the RFID tag. **Objects** already encoded to UTF-8 encoding rules should be qualified with the **Compact Parameter** value (2) so that subsequent reading of the **Object** will clearly indicate that it shall be processed through a UTF-8 decoder for final presentation to the application receiving the data. The **Compact-Parameter** value (0) should only be used if there is an over-riding reason not to compact the data. Reasons for this include prior compaction to application rules, or if the **Object** has been encrypted. In both these cases, although compaction could be possible and the decode process would fully restore the **Object**, the originally defined parameter value 0 or 2 would be lost and the receiving application could fail to undertake subsequent processing.

An alternative to the basic compaction is for a set of **Objects** to be encoded to the rules of Packed Objects. This can only be done if the application administrators choose to adopt the scheme by creating an index table, to which source references should be provided on the register of data constructs. Then for each RFID tag to be encoded to the Packed Object rules the **Compact-Parameter** value (3) is applied to the entire set of **Objects** to be encoded.

Similarly, if data is to be encoded to the rules of Tag Data Profiles the **Compact-Parameter** value (4) is applied to the entire set of **Objects** to be encoded.

During the decode process, the following response codes are applied based on the input conditions:

- If the command argument was **Application-Defined** (0), a compliant Data Processor undertook no compaction and applied the relevant compact code to the encoding on the tag. As the decoder cannot interpret the encoded bytes, the response is **Application-Defined** (0).
- If the command argument was **Compact** (1) a compliant Data Processor applied the correct compact code to the encoding on the tag. As the decoder can interpret the encoded bytes, the response is **De-Compacted-Data** (15).
- If the command argument was **UTF8-Data** (2) a compliant Data Processor undertook no compaction and applied the relevant compact code to the encoding on the tag. As the decoder cannot interpret the encoded bytes, the response is **UTF8-Data** (2).
- If the command argument was **Pack-Objects** (3) a compliant Data Processor encoded to the rules as defined for the particular **Object** as defined by the ID Table specified. The decoder uses the same ID table to interpret the encoded byte and returns each data **Object** with the response **De-Compacted-Data** (15).
- If the command argument was **Tag-data-Profile** (4) a compliant Data Processor encoded to the rules as defined for the particular **Object** as defined by the ID Table specified. The decoder uses the same ID table to interpret the encoded byte and returns each data **Object** with the response **De-Compacted-Data** (15).
- If the command argument was **Monomorphic-UII** (5) a compliant Data Processor encoded to the rules as defined for the particular **Object** as defined by the AFI and the Data Constructs register. The decoder, on recognition of the AFI uses the compaction rule declared on the Data Constructs register and returns the data **Object** with the response **De-Compacted-Monomorphic-UII** (14). It also returns the **Object-Identifier** as defined by the Data Constructs register.

NOTE Although the decoder is required to look up the compaction and de-compaction rules, as declared by the AFI from the Data Constructs register, responses from reading data do not include the AFI. This response argument is necessary to ensure that additional matching procedures are can be carried out.

7.3.6 Object-Lock

The command argument **Object-Lock** requires the Data Processor to arrange the encoded data set (encoded **Object-Identifier** or **Relative-OID**, **Object**, Precursor and other associated syntax components) in such a way that all of the associated bytes can be stored and locked in a block aligned manner in the Logical Memory Map. The locking process shall have the effect of making the bytes so processed permanently encoded on the RFID tag, or only capable of being unlocked in some types of RFID tag with the use of relevant passwords.

7.4 Other command arguments

7.4.1 Access-Password

The command argument **Access-Password** requires the Data Processor to pass this to the interrogator so that the **Access-Password** in the command is matched with that on the RFID tag. A match results in additional actions on the RFID tag being permitted. A mismatch results in the air interface rejecting the command.

7.4.2 Additional-App-Bits

The command argument **Additional-App-Bits** is used to extend the search criteria of **Object-Identifier** data encoded in the UII memory of a segmented memory RFID tag. The **Additional-App-Bits** are added as a suffix to the **AFI** and **DSFID**.

7.4.3 AFI-Lock

The command argument **AFI-Lock** is used to determine whether the **AFI** is to be locked or not. This is a BOOLEAN argument. If set to TRUE, the interrogator shall lock the **AFI** to ensure that the particular RFID tag can only be used in the way prescribed. The locking process shall have the effect of making the bytes so processed permanently encoded on the RFID tag, or only capable of being unlocked in some types of RFID tag with the use of relevant passwords.

NOTE In some tag types the **AFI** is part of a contiguous string with the encoded data sets. In such cases it cannot be locked independently.

7.4.4 Append-To-Existing-Multiple-Record

The command argument **Append-To-Existing-Multiple-Record** is a BOOLEAN argument. If TRUE, the data objects are to be appended to an existing multiple record, subject to various checks by the Data Processor. If FALSE, then a new multiple record shall be created.

7.4.5 Application-Defined-Record-Capacity

The command argument **Application-Defined-Record-Capacity** is a BOOLEAN argument that if FALSE determines that the Data Processor automatically determines that the capacity for the given multiple record is simply based on the encoded data and then increased to be block aligned. If set to TRUE, then the application needs to set the size of memory assigned to the record using the **Record-Memory-Capacity** argument.

7.4.6 Avoid-Duplicate

The command argument **Avoid-Duplicate** is used to ensure that the **Object-Identifier** is not already encoded in the Logical Memory Map. This is a BOOLEAN argument. If set to TRUE, the interrogator shall verify all **Object-Identifiers** in the Logical Memory Map. If the **Object-Identifier** already exists, the write command is aborted and the **Completion-Code** value **Duplicate-Object** (10) is returned. If set to FALSE, the interrogator shall write the **Object** without any verification.

7.4.7 Battery-Assist-Indicator

The command argument **Battery-Assist-Indicator** is used by the application that the RFID tag supports a battery assist feature. The argument is generally used when the air interface protocol does not have a feature to support such an indication.

7.4.8 Block-Align

The command argument **Block-Align** is used to define whether **Objects** written to a tag shall be aligned to the beginning of a block. This is a BOOLEAN argument. If set to TRUE, the interrogator shall align each **Object** written to a tag on a block boundary as determined by the specific tag architecture or manufacture.

7.4.9 Block-Align-Packed-Object

The command argument **Block-Align-Packed-Object** is used to define whether **Packed-Objects** written to a tag shall be aligned to the beginning of a block. This is a BOOLEAN argument. If set to TRUE, the interrogator shall align each **Packed-Object** written to a tag on a block boundary as determined by the specific tag architecture or manufacture.

7.4.10 Check-Duplicate

The command argument **Check-Duplicate** is used to invoke a particular process defined by the command (e.g. read, or delete). This is a BOOLEAN argument.

If set to TRUE, the interrogator shall verify all **Object-Identifiers** in the Logical Memory Map. The command shall only be completed if there is no duplicate present. Otherwise, the **Completion-Code** value **Duplicate-Object** (10) is returned.

If the **Check-Duplicate** flag is set to FALSE, the interrogator shall act upon the first occurrence of the **Object-Identifier**, associated **Object** and precursor. In this case, a duplicate **Object-Identifier** and associated **Object** (possibly with a different value) and precursor might be present.

7.4.11 Data-CRC-Indicator

The command argument **Data-CRC-Indicator** in a command instructs the Data Processor to add a CRC-16 either to each individual data set or to the entire encoded data, or to both. The argument **Data-CRC-Indicator** in a response indicates that the Data Processor validated the CRC-16, stripped this from the response and transferred the interpreted **Object-Identifier** and associated **Object** to the application.

7.4.12 Data-Length-Of-Record

This response argument provides the size of the encoded Multiple Record in terms of the write block size, as encoded in EBV-8 format in the record preamble.

7.4.13 Delete-MR-Method

The command argument **Delete-MR-Method** is used by the application to invoke one of two methods for the interrogator to delete a multiple record. This command argument is presented as an Integer value, and the following codes apply:

- 0 Mark record as deleted
- 1 Physically delete the record

The process details are defined in 10.27.1.

7.4.14 Directory-Length-EBV8-Indicator

The command argument **Directory-Length-EBV8-Indicator** is used by the application to ensure that sufficient space is available in the MR-header for the Data Processor to encode the actual length of the directory. This command argument is presented as an integer value and the following codes apply:

- 1 Single byte EBV-8, which can support a directory of up to 127 blocks.
- 2 Double byte EBV-8, which should be used when the Logical Memory is large and / or many records are expected to be encoded on it. This can support a directory size of up to 16383 blocks.

The value 2 is used for a directory that might originally be small in size, but is expected to increase in size at some future time.

7.4.15 DSFID-Lock

The command argument **DSFID-Lock** (previously **storageFormatLock**) is used to determine whether the **DSFID** is to be locked or not. This is a BOOLEAN argument. If set to TRUE, the interrogator shall lock the **DSFID** to ensure that the particular RFID tag can only be used in the way prescribed. The locking process shall have the effect of making the bytes so processed permanently encoded on the RFID tag, or only capable of being unlocked in some types of RFID tag with the use of relevant passwords.

NOTE In some tag types the **DSFID** is part of a contiguous string with the encoded data sets. In such cases it cannot be locked independently.

7.4.16 DSFID-Pad-Bytes

The command argument **DSFID-Pad-Bytes** specifies the number of bytes that the application wants to provide in an **Extended-DSFID** in addition to those that are created by the Data Processor. For example this could be to provide the encoding space for the length of encoded data to be added and updated in the future. When included in a response, additional pad bytes might be added by the Data Processor to align on a lock-block boundary if the **Extended-DSFID** is to be locked, or if the first **Data-Set** is to be locked.

7.4.17 Editable-Pointer-Size

The command argument **Editable-Pointer-Size** is used to specify that a Packed Object created explicitly or implicitly through a **Write-Objects** command shall allow subsequent modifications to its contents or structure. This command argument is presented as a non-zero integer value that indicates that the created Packed Object shall be made editable by adding an optional addendum subsection to the end of the Object Info section of the Packed Object. The pointer(s) to the Addendum Packed-Object shall be the number of bits specified in this command argument. The default value for this argument is zero, indicating that no addendum subsection is present, and that therefore the Packed Object is not editable.

7.4.18 Encoded-Memory-Capacity

This response argument provides the size reserved for a Multiple Record in terms of the write block size, as encoded in EBV-8 format in the record preamble.

7.4.19 EPC-Code

The command argument **EPC-Code** represents any of the unique codes defined by the EPCglobal Tag Data Standard. The structure of each **EPC-Code** is self-declaring from the value of its 8-bit header. Although some of the codes do not align on an 8-bit boundary, these need to be rounded to 8-bit bytes for processing through the Data Processor, and be rounded to 16-bit words for encoding on the associated RFID tag.

7.4.20 Full-Function-Sensor-Indicator

The command argument **Full-Function-Sensor-Indicator** is used by the application that the RFID tag supports a full-function sensor. The argument is generally used when the air interface protocol does not have a feature to support such an indication.

7.4.21 Hierarchical-Identifier-Arc

The response argument **Hierarchical-Identifier-Arc** is an integer value, converted from the EBV-8 value encoded in the Multiple Records preamble from a hierarchical record.

7.4.22 Identifier-Of-My-Parent

The command and response argument **Identifier-Of-My-Parent** is used for a multiple record that is part of a hierarchy, and has the value of the hierarchical code of that record.

7.4.23 Identify-Method

The command argument **identify-Method** is used to define whether all, or some, of the RFID tags belonging to the particular selected **AFI** in the operating area shall be identified. This command argument is presented as an integer value and the following codes apply:

0	Inventory-All-Tags
1	Inventory-At-Least

- 2 **Inventory-No-More-Than**
- 3 **Inventory-Exactly**
- 4 to 15 reserved for future definition

For every argument including **Inventory-All-Tags**, it is necessary to specify the number of RFID tags to be read using the **Number-Of-Tags** (see 7.4.43). More precise advice is provided in **Inventory-Tags** command (see 10.3).

7.4.24 ID-Type

The command argument **ID-Type** is used to specify that a **Packed-Object** created explicitly or implicitly through a **Write-Objects** command shall be instantiated as type of ID List or ID Map. This command argument is presented as an integer value and the following codes apply:

- 0 **ID List**
- 1 **ID Map**
- 2 to 15 Reserved for future definition

7.4.25 Instance-Of-Arc

The response argument **Instance-Of-Arc** is an integer value, converted from the EBV-8 value encoded in the Multiple Records preamble.

7.4.26 Kill-Password

The command argument **Kill-Password** requires the Data Processor to pass this to the interrogator so that the **Kill-Password** in the command is matched with that on the RFID tag. A match results in the RFID tag function being permanently disabled. A mismatch results in the air interface rejecting the command.

7.4.27 Length-Of-Mask

The command argument **Length-Of-Mask** is used in conjunction with the command arguments **Pointer** and **Tag-Mask** to define the search criteria of EPCglobal data encoded in the UII memory of a segmented memory RFID tag.

7.4.28 Lock-Directory-Entry

The command argument **Lock-Directory-Entry** is BOOLEAN and if set to TRUE the interrogator shall lock the directory entry for the particular Multiple Record.

7.4.29 Lock-Multiple-Records-Header

The command argument **Lock-Multiple-Records-Header** is used to determine if all, some, or none of the MR-header is locked. This command argument is presented as an integer value and the following codes apply:

- 0 Not locked
- 1 Completely locked
- 2 the number of records field remains unlocked, the preceding fields are locked
- 3 the Data Length of the Directory field remains unlocked, all other fields including the Number of Records field are locked
- 4 the Data length of the directory field and the Number of Records field remain unlocked, all other fields are locked

7.4.30 Lock-Record-Preamble

The command argument **Lock-Record-Preamble** is BOOLEAN and if set to TRUE the interrogator shall lock the fields in the preamble.

7.4.31 Lock-UII-Segment-Arguments

The command argument **Lock-UII-Segment-Arguments** is used to define which component parts of the UII segment of an ISO/IEC 18000-6 Type D tag shall be locked. The argument takes precedence over lock argument elsewhere in the associated command.

7.4.32 Max-App-Length

The command argument **Max-App-Length** is used to define the maximum compacted length of an encoded data set. It is used in commands that need to constrain air interface read commands for faster transactions, for example to read a UII encoded in the first data set on the RFID tag.

7.4.33 Memory-Bank

The command argument **Memory-Bank** is used to define which part of a segmented memory tag on which data is to be encoded. The Data Processor then invokes rules that are specific to the **Memory-Bank** to achieve a correctly encoded tag.

7.4.34 Memory-Bank-Lock

The command argument **Memory-Bank-Lock** is used to define which part of a segmented memory tag is to be locked.

7.4.35 Memory-Length-Encoding

The command argument **Memory-Length-Encoding** in a command instructs the Data Processor to encode either memory capacity or the length of encoded data, or two both, in terms of the number of blocks. The argument **Memory-Length-Encoding** in a response indicates that the Data Processor calculated the relevant length and encoded this as part of the extended DSFID.

7.4.36 Memory-Segment

The command argument **Memory-Segment** is used to define which part of a segmented memory tag on which data is to be encoded. It is similar in function to the **Memory-Bank** argument (see 7.4.33), except that the RFID tag for which this argument is appropriate can address multiple segments in the same air interface transactions.

7.4.37 Memory-Type

The command argument **Memory-Type** is used to define which memory structure is intended for encoding a **Monomorphic-UII**, effectively instructing the Data Processor which of the optional and conditional arguments and processes to apply to the encoding process.

7.4.38 Multiple-Records-Directory-Length

The response argument **Multiple-Records-Directory-Length** is an EBV-8 value.

7.4.39 Multiple-Records-Features-Indicator

The command and response argument **Multiple-Records-Features-Indicator** is a bit map with the following structure:

- Bit 8 with the value 1₂ indicates that all the records use the same **Access-Method**. If this bit = 0 then bits 7 to 4 = 0000₂
- Bits 7 to 4 identify the particular **Access-Method** if this is applied consistently. If bit 8 = 0, the currently permissible string for bits 7 to 4 are:

0000 indicates that all records use the **No-Directory Access-Method**
 0001 is not permitted
 0010 indicates that all records use the **Packed-Objects Access-Method**
 0011 indicates that all records use the **Tag-Data-Profile Access-Method**
 0100 is not permitted
 0101 to 1111 is currently reserved for additional **Access-Methods**

- Bit 3 with the value 1 indicates that some of the records are in a hierarchical relationship with other records.
- Bit 2 identifies whether the number of records field is fully maintained (e.g. updated each time a new record is added) or whether the number of records might not be the correct current value. The value 1 indicates that the number of records is correct, the value 0 indicates that the number of records field might be incorrect.
- Bit 1 with the value 1 indicates that there is an additional byte for indicating additional features. Currently this is RFU and therefore set to 0.

7.4.40 NSI-Bits

The command argument **NSI-Bits** defines a 9-bit code defined by EPCglobal used as a prefix to the **EPC-Code** when encoded on the associated RFID tag.

7.4.41 Number-In-Data-Element-List

The command argument **Number-In-Data-Element-List** is only used in a hierarchical record that is a data element list. The value is the count of the number of instances-of the data element being encoded.

7.4.42 Number-Of-Records

The response argument **Number-Of-Records** is used in one of two ways. If bit 2 of the **Multiple-Records-Features-Indicator** equals:

- 0 then this field is not maintained and the value of **Number-Of-Records** should be zero, but any other value should be ignored, for example if some record count was initially maintained but later this function was decided to be stopped.
- 1 then this field is fully maintained as new records are added. A zero value for the **Number-Of-Records** is encoded when the MR-header is created.

7.4.43 Number-Of-Tags

The command argument **Number-Of-Tags** is used to define a limit on the particular **Identify-Method** specified. It is an integer value in the range 0 to 65535.

7.4.44 Objects-Offsets-Multiplier

The command argument **Object-Offsets-Multiplier** defines the size of the memory in bits that is requested for the storage of object offsets when the **Directory-Type** parameter in the **Packed-Objects-Constructs** argument is **Packed-Object** offset. The implementation shall use the parameter for proper sizing of the AuxMap structure in the Packed Object.

7.4.45 Packed-Object-Directory-Type

The command argument **Packed-Object-Directory-Type** defines whether the Packet Object is a directory, and in this case which type of Packed Object directory has been constructed. A Presence/Absence directory provides a bit map of the ID Values encoded on any of the Packed Objects on the RFID tag, but no indication of where the data is encoded. The index field directory adds the ordinal value of the Packed Object containing the specific ID Value. For example a 3-bit index field identifies which of eight Packed Objects to access. An offset directory provides the starting address of the Packed Object containing a particular ID Value.

A null directory (i.e. the provision of space for a directory is created by setting the pointer allocation in this argument combined an non-zero value for the **PO-Directory-Size** argument.

If a Packed Object is not a directory, then a zero value is set for this argument.

7.4.46 Password

The command argument **Password** defines the byte string that represents a code value, qualified by the **Password-Type**, in a command that is required to be matched with the similarly defined code on the RFID tag. A match provides permissions to carry out additional actions with the RFID tag. A mismatch results in any associated command to be rejected.

7.4.47 Password-Type

The command argument **Password-Type** qualifies the value of the **Password**, used to provide permissions to carry out additional actions with the RFID tag.

7.4.48 PO-Directory-Size

The command argument **PO-Directory-Size** defines the size of the (null) directory pointer created with the Packed Object. This command argument is presented as a non-zero integer value that shall be used for the size of the (null) directory pointer encoded in the Packed Object.

7.4.49 PO-Index-Length

The command argument **PO-Index-Length** is used to specify the size of the POindex Length for the AuxMap structure to be created when the Packed Object **Directory-Type** parameter is **Packed-Object** index field.

7.4.50 Pointer

The command argument **Pointer** is used in conjunction with the command arguments **Length-Of-Mask** and **Tag-Mask** to define the search criteria of EPCglobal data encoded in the UII memory of a segmented memory RFID tag.

7.4.51 Pointer-To-Multiple-Records-Directory

The command and response argument **Pointer-To-Multiple-Records-Directory** is used to define the highest block number at the start address of the directory. This is defined as an EBV-8 value. If a directory is not initially encoded, then a fixed length EBV-8 string of the same length as required for the start point of the directory shall be encoded with a value zero. The directory is encoded in reverse block sequence so that the next subsequent block of the directory is at the next lower address. The start of the directory is not necessarily

the highest address in Logical Memory, because other hardware features of the tag might be specified to be located there. It will be necessary to invoke the air interface commands that define the memory mapping to determine the start address of the directory.

7.4.52 Read-Record-Type

The command argument **Read-Record-Type** is used to identify various logical structures from the RFID tag. This command argument is presented as an integer value and the following codes apply:

- 0 **Read-Multiple-Records-Header**
- 1 **Read-Multiple-Records-Header-Plus-1st-Preamble**
- 2 **Read-Multiple-Records-Directory**
- 3 **Read-Preamble-Specific-Multiple-Record**
- 4 **Read-All-Record-OIDs-Specific-Record-Type**
- 5 **Read-OIDs-Specific-Multiple-Record**
- 6 **Read-All-Objects-Specific-Multiple-Record**
- 7 **Read-Multiple-Objects-Specific-Multiple-Record**
- 8 **Read-1st-Objects-Specific-Multiple-Record**
- 9 **Read-Data-Element-List-Specific-Multiple-Record**

If **Read-Multiple-Records-Header** is selected, the Data Processor returns the interpretation of the MR-header in the **Multiple-Records-Header-Structure** argument.

If **Read-Multiple-Records-Header-Plus-1st-Preamble** is selected, the Data Processor returns the interpretation of the MR-header in the **Multiple-Records-Header-Structure** argument and the interpretation of the first record's preamble in the **Multiple-Records-Preamble-Structure** argument.

If **Read-Multiple-Records-Directory** is selected, the Data Processor returns the interpretation of the multiple records directory in the **Multiple-Records-Directory-Structure** argument.

The **Read-Record-Type** codes 3 to 9 require the use of one or more **Object-Identifiers** in the command for the Data Processor to invoke the relevant processes. Three very specific formats of **Object-Identifier** are applied to multiple records:

- For a multiple record that is not part of a hierarchy, the structure is: **1.0.15961.401.{Data-Format}.{sector identifier}.{record type}.{instance-of}.{Relative-OID of data element}**
- For a multiple record that is part of a hierarchy, but not a data element list, the structure is: **1.0.15961.402.{Data-Format}.{sector identifier}.{record type}.{hierarchical id}.{Relative-OID of element}**
- For a multiple record that is a data element list, the structure is: **1.0.15961.403.{Data-Format}.{sector identifier}.{record type}.{hierarchical id}.{data element}**

NOTE This object identifier calls for the response to include all the list element number

The first two **Object-Identifier** structures apply to **Read-Record-Type** codes 3 to 8. The third listed **Object-Identifier** structure only applies to **Read-Record-Type** codes 3, 4 and 9.

If **Read-Preamble-Specific-Multiple-Record** is selected, the command shall include a single **Object-Identifier** in the **Read-Objects List** that is certainly defined down to the record type arc, and the instance-of arc (if applicable) or the hierarchical id arc (if applicable). The Data Processor returns the interpretation of the record's preamble in the **Multiple-Records-Preamble-Structure** argument.

Selecting the **Read-All-Record-Olds-Specific-Record-Type** is useful for identifying a series of history records of the same type or a set of records in a hierarchy of the same type. In this case the command shall include a single **Object-Identifier** in the **Read-Objects List** that is only defined down to the record type arc. The Data Processor returns the list of **Object-Identifiers** one layer lower in the **Read-Olds-Response-List**, i.e. either with the set of instance-of arcs or with the set of hierarchical id arcs.

If **Read-Olds-Specific-Multiple-Record** is selected, the command shall include a single **Object-Identifier** in the **Read-Objects List** that is certainly defined down to the record type arc, and the instance-of arc (if applicable) or the hierarchical id arc (if applicable). The Data Processor returns the list of **Object-Identifiers** encoded within the record in the **Read-Olds-Response-List**. This does not apply to Data Element lists.

If **Read-All-Objects-Specific-Multiple-Record** is selected, the command shall include a single **Object-Identifier** in the **Read-Objects List** that is certainly defined down to the record type arc, and the instance-of arc (if applicable) or the hierarchical id arc (if applicable). The Data Processor returns the list of **Object-Identifiers** and **Objects** encoded within the record in the **Read-Objects-Response-List**. This does not apply to Data Element lists.

If **Read-Multiple-Objects-Specific-Multiple-Record** is selected, the command shall include the nominated **Object-Identifiers** that are defined down to the specific data element in the **Read-Objects List**. The Data Processor returns the list of nominated **Object-Identifiers** and **Objects** encoded within the record in the **Read-Objects-Response-List**. This does not apply to Data Element lists.

If **Read-1st-Objects-Specific-Multiple-Record** is selected, the command shall include the nominated **Object-Identifier(s)** that are defined down to the specific data element(s) in the **Read-Objects List**. The Data Processor returns the list of nominated **Object-Identifiers** and **Objects** encoded up to the **Max-App-Length** within the record in the **Read-Objects-Response-List**. This does not apply to Data Element lists.

If **Read-Data-Element-List-Specific-Multiple-Record** is selected, the command shall include a single **Object-Identifier** in the **Read-Objects List** that is defined down to the data element. The Data Processor first checks that the record is a Data Element list. If so, it uses the rules of the **Access-Method** to re-construct the **Object-Identifier** down to the list element number as encoded in the data element list and returns this and associated **Objects** encoded within the record in the **Read-Objects-Response-List**.

7.4.53 Read-Type

The **Read-Type** is a command argument used to identify the number and location of **Object-Identifiers** in a read command. This command argument is presented as an integer value and the following codes apply:

0	Read-1st-Objects
1	Read-Multiple-Objects
2	Read-All-Objects
3	Read-Monomorphic-UII
4 to 15	Reserved for future definition

If **Read-1st-Objects** is selected, then the command needs to include the value for the **Max-App-Length** code, which is equivalent to the number of bytes intended to be read into the application. The argument is so structured – and therefore differs from ISO/IEC 15961:2004 – to read a sequence of **Object-Identifiers**, rather than the one only in the first position. If the **Read-Type** is set for **Read-Multiple-Objects**, this can apply to one or more **Object-Identifiers**.

If **Read-Monomorphic-UII** is selected, it instructs the Data Processor to carry out additional checks with the ISO/IEC Data constructs register to enable the proper process on the command.

7.4.54 Record-Memory-Capacity

The command argument **Record-Memory-Capacity** is used in commands to write a multiple record to indicate the amount of memory (in terms of write blocks) to assign, usually to enable the record to have additional data elements added. It is only used if the application needs to over-ride the automatic sizing by the Data Processor, when the **Application-Defined-Record-Capacity** argument is set to TRUE.

7.4.55 Record-Type-Arc

The response argument **Record-Type-Arc** is an integer value, converted from the EBY-8 value encoded in the Multiple Records preamble.

7.4.56 Record-Type-Classification

The command and response argument **Record-Type-Classification** is a bit string that identifies the class of record being encoded. The following codes apply:

000	stand-alone record, with an instance-of arc = 0
001	stand-alone record, with an instance-of arc >0
010	hierarchical record, top level
011	hierarchical record, has both parent and child(ren)
100	hierarchical record, data element list
101	other hierarchical record, no further children (never applies to a data element list)
110	Not relevant to this command (because it is associated with deleted records)
111	reserved

7.4.57 Sector-Identifier

The **Sector-Identifier** is used in the MR-header either to indicate the true **Sector-Identifier** for all records in the Logical Memory, or to signal that the true value varies between records and the true value is encoded in the record. This command and response argument is presented as an integer value and the following codes apply:

- 0 The true **Sector-Identifier** is encoded in each record and can vary between records
- 1 This is used for closed system applications
- 2 This indicates that the record type has a value equal to the **Relative-OID** of the first data element encoded on the record. For example if the first data element has a **Relative-OID** = 7 for a product code, then the Record type = 7
- >2 This is the **Sector-Identifier** assigned by the administrators of the data dictionary to a sector to manage its own allocation record types.

NOTE If all the records have the same **Sector-Identifier**, the non-zero value is part of the **Object-Identifier** structure in a command to write or read a record, but has to be included separately for the **Configure-Multiple-Records-Header** command.

7.4.58 Simple-Sensor-Indicator

The command argument **Simple-Sensor-Indicator** is used by the application that the RFID tag supports a simple sensor. The argument is generally used when the air interface protocol does not have a feature to support such an indication.

7.4.59 Start-Address-Of-Record

This response argument provides the address of first byte of the encoded Multiple Record in terms of the write block size, as encoded in EBV-8 format in the multiple records directory.

7.4.60 Tag-Data-Profile-ID-Table

The **Tag-Data-Profile-ID-Table** is a command argument used to identify the particular **Tag-Data-Protocol** that contains the compaction and formatting rules. The specific rules are essential to encode any data for a given application using this **Access-Method**.

7.4.61 Tag-Mask

The command argument **Tag-Mask** is used in conjunction with the command arguments **Length-Of-Mask** and **Pointer** to define the search criteria of EPCglobal data encoded in the UII memory of a segmented memory RFID tag.

7.4.62 Update-Multiple-Records-Directory

The command argument **Update-Multiple-Records-Directory** is BOOLEAN, but the processing on the Logical Memory by the Data Processor needs to take into account whether a directory already exists. The following states and processes apply:

- If a directory pre-exists and the command argument is set to TRUE, the directory is fully updated, including any previously missed directory entries.
- If a directory pre-exists and the command argument is set to FALSE, the argument is ignored and the directory is fully updated, including any previously missed directory entries.
- If no directory exists and the command argument is set to TRUE, the directory is created and fully updated, including any previously missed directory entries.
- If no directory exists and the command argument is set to FALSE, no directory is created.

7.4.63 Word-Count

The command argument **Word-Count** is used in conjunction with **Word-Pointer** to define the number of encoded bytes to be read from a segmented memory RFID tag, without any processing to be undertaken by the Data Processor.

7.4.64 Word-Pointer

The command argument **Word-Pointer** is used in conjunction with **Word-Count**, and defines the start location of the memory of a segmented memory RFID tag from which an encoded byte string is to read.

7.5 Command-related field names

In addition to the command arguments (7.4), the following field names are used in the commands and responses.

7.5.1 Data-Set

The field name **Data-Set** refers to the contiguous byte string from the Precursor, the **Object-Identifier** to the final byte of the compacted **Object** on an RFID tag.

7.5.2 Identities

The field name **Identities** is a command response field that represents a list of the identified **Singulation-Ids**.

7.5.3 Length-Lock Byte

The field name **Length-Lock Byte** is a command response field that identifies the length of the encoded data in an item-related segment, and also provides information about the lock status of pages on the RFID tag.

7.5.4 Length-Of-Encoded-Data

The field name **Length-Of-Encoded-Data** is a command response field that is part of the Extended **DSFID** and identifies the length of the encoded data in terms of blocks.

7.5.5 Lock-Status

The field name **Lock-Status** is a command response field that identifies whether an encoding packet (e.g. a **Data-Set**) is locked or not locked.

7.5.6 Logical-Memory-Map

The field **Logical-Memory-Map** is a command response field that represents the complete, but un-decoded, byte string that was read from the RFID tag.

7.5.7 Memory-Capacity

The field name **Memory-Capacity** is a command response field that is part of the Extended **DSFID** and identifies the length of the encoded data in terms of blocks.

7.5.8 Module-OID

The field **Module-OID** identifies an individual command or response module with a full **Object-Identifier** as defined in 7.3.2.

7.5.9 Number-Of-Tags-Found

The field name **Number-Of-Tags-Found** is a command response field that returns the actual number of RFID tags observed that met the criteria. If the **Identify-Method** argument is set to **Inventory-No-More-Than**, then the response value **Number-Of-Tags-Found** can be a lower number.

7.5.10 PO-ID-Table

The field name **PO-ID-Table** is a command field that identifies the specific table for the Data Processor to use as the source of detailed encoding rules.

7.5.11 Protocol-Control-Word

The field name **Protocol-Control-Word** is a command response field that returns this 16-bit value in a response from the UII memory of a segmented memory RFID tag. The **Protocol-Control-Word** contains bit-based data that identifies other features supported on the RFID tag.

7.5.12 Read-Data

The field **Read-Data** returns an un-decoded byte string from a segmented memory RFID tags.

7.6 Data security

The data (**Object**) may be made secure by the use of some form of encryption. This shall be applied prior to the **Object** being transferred to the Data Processor. The decryption process shall also be applied to the **Object** after it has been transferred to the application. As such, all the processes are transparent to this part of ISO/IEC 15961 and ISO/IEC 15962.

The following features can be provided:

- Data security - This allows only authorised users the ability to "see" the true data. The sender will need to provide authorised users with the decryption algorithm and key. Other users can see the raw octet string, but this will be meaningless.
- Data integrity - This uses the decrypted data to identify whether the data has been modified by others prior to being read by an authorised user.
- Data validity - This uses encryption processes to reduce the risk of data from a legitimate source being copied by an unauthorised agent into another RFID tag, and being passed off as the original source RFID tag.

Additional advice is provided in Annex D.

8 Data flows and processes to the air interface

Various processes are required to format the RFID tag, to write data to it, to read from it, to modify data etc. These are defined in the sub-clauses that follow. All the processes to write and add data will be described. Where the read process is the inverse of the write process, this will be described briefly, otherwise a further description will be provided.

8.1 Establishing communications between the application and the RFID tag

The Data Processor does not communicate directly with the RFID tag (see Figure 1 — Schematic of protocol layers for an implementation of RFID for item management and Figure 2 — Logical functions and interfaces of ISO/IEC 15962 with other RFID system components), but does this through the Tag Driver. The Data Processor requires specific system information based on the configuration of the RFID tag (see 8.1.2). This is to set parameters of the Logical Memory to represent correctly the RFID tag memory and to enable communication to and from the Tag Driver. To achieve this, air interface services shall be provided to the Data Processor via the Tag Driver to establish communications (see 8.1.1).

A number of these parameters need to be known to, or requested by, the Data Processor. Effectively, this procedure is used to configure the RFID tag initially, to re-configure it if required, and to establish a communications link while the data transaction is open.

8.1.1 Air interface services

This part of ISO/IEC 15961 is open-ended with respect to the fact that new types of RFID tag may be added to ISO/IEC 18000 and leave the Data Processor unaltered. To achieve this, some basic presumptions are made about the types of RFID tag in ISO/IEC 18000.

- Application memory is an integer number of bytes.

NOTE the term application memory is used in this sub-clause as a generic name for the area of RFID tag memory available for user data (this is sometimes called user memory in ISO/IEC 18000) and any separately addressable memory (e.g. for the UII).

- Application memory shall be organised in blocks. These shall be fixed size and be of one or more bytes.
- The individual blocks shall each be accessible by read and/or write.

NOTE This applies to the basic function, additional features may be used to restrict access to authorised users.

- In addition to the requirements (above) relating to the memory, there shall be a reliable mechanism for writing and reading to and from the application memory.

The RFID tag shall have a mechanism for storing the system information (see 8.1.2), including the ability to write and read the component elements.

The technical details of the air interface services are provided in ISO/IEC 15962.

8.1.2 System information

The system information shall consist of the following elements that need to be transferred across the application interface and air interface, and are therefore part of this part of ISO/IEC 15961 and ISO/IEC 15962:

- Singulation-Id** – (see 7.2.1)
- AFI** – (see 7.2.2)
- DSFID** – (see 7.2.3), which itself consists of:
 - Access-Method** – (see 7.2.4)
 - Data-Format** – (see 7.2.5)

The system information shall also consist of the following elements that only need to transfer between the RFID tag and the Data Processor to configure the Logical Memory Map, and are therefore only part of ISO/IEC 15962:

- physical block size
- number of blocks

8.2 Application system services

The application system shall provide the following:

- **Object Identifier** - (see 7.3.2)
- **Object** - (see 7.3.4)
- **Compact-Parameter** - (see 7.3.5)
- **Object-Lock** - (see 7.3.6)

These are incorporated into the definitions of particular application commands and responses, and never transferred without the supporting commands.

9 Command-Codes, Completion-Codes, and Execution-Codes

Application commands are used to instruct the Data Processor and the interrogator to execute specific functions. They are also applied to the processing of the application data to achieve efficient encoding. The responses from the Data Processor include requested data and also information about actions undertaken

and errors found. Each command/ response pair has its abstract syntax presented as modules. Each module is defined in a manner that enables the command to be invoked independently of any other command.

So that the commands and responses can easily be incorporated into the transfer syntax, code values have been assigned in this part of ISO/IEC 15961 to the final arc of the command and response modules (see 9.1). Also, **Completion-Codes** (see 9.2) and **Execution-Codes** (see 9.3) are assigned to the responses. The sources of definitions of arguments and fields that apply to each command are given, including those that only apply to the command and response modules (see 7.4 and 7.5).

In the processing of a command, an error might be detected. The command shall be aborted and no data is transferred to or from the RFID tag in which the error is detected. This is possible because processing is done in the Logical Memory. Although other error conditions might be present, the first problem identified is the only one reported. The appropriate **Completion-Code** or **Execution-Code** is returned. In the case where the command addresses multiple RFID tags, all RFID tags processed prior to the one with the detected error should be processed. The detection of the error aborts all subsequent processing.

The following sub-clauses define all the application commands and responses the are supported by this edition of this part of ISO/IEC 15961. In addition to the basic syntax, these sub-clauses will also describe the function and purpose of specific command arguments and responses. The commands and responses are grouped logically together. Within this clause, the current appropriate abstract syntax is shown.

Annex E shows the original 16 modules using the ASN.1 abstract syntax, and with the terminology used in ISO/IEC 15961:2004. Some of these modules have direct equivalents in the current format, others have been merged to create new modules and this will be described in the relevant sub-clauses. Annex F shows an example of the original transfer encoding of a command and response.

9.1 Final arc values of the command and response modules

Each command and response module shall be identified by an **Object Identifier**. The common root for commands is **{iso(1) standard(0) rfid-data-protocol(15961) commandModules(126)}**. The common root for responses is **{iso(1) standard(0) rfid-data-protocol(15961) commandResponses(127)}**. The final arc of each pair of command and response modules shall have the same value, effectively a **Relative-OID**. The final arc shall be specific to the command/response pair and the following final arcs are specified:

- | | |
|----|--|
| 1 | Configure-AFI (in ISO/IEC 15961:2004 as configureAfi) |
| 2 | Configure-DSFID (in ISO/IEC 15961:2004 as configureStorageFormat) |
| 3 | Inventory-Tags |
| 4 | (only in ISO/IEC 15961:2004 as addSingleObject) |
| 5 | Delete-Object |
| 6 | Modify-Object |
| 7 | (only in ISO/IEC 15961:2004 as readSingleObject) |
| 8 | ReadObject-Identifiers (in ISO/IEC 15961:2004 as readObjectIds) |
| 9 | (only in ISO/IEC 15961:2004 as readAllObjects) |
| 10 | Read-Logical-Memory-Map |
| 11 | (only in ISO/IEC 15961:2004 as inventoryAndReadObjects) |
| 12 | Erase-Memory |
| 13 | Get-App-Based-System-Info |
| 14 | (only in ISO/IEC 15961:2004 as addMultipleObjects) |
| 15 | (only in ISO/IEC 15961:2004 as readMultipleObjects) |
| 16 | (only in ISO/IEC 15961:2004 as readFirstObject) |
| 17 | Write-Objects |
| 18 | Read-Objects |
| 19 | Write-Objects-Segmented-Memory-Tag |
| 20 | Write EPC-Ull |
| 21 | Inventory-ISO-Ullmemory |
| 22 | Inventory-EPC-Ullmemory |
| 23 | Write-Password-Segmented-Memory-Tag |
| 24 | Read-Words-Segmented-Memory-Tag |
| 25 | Kill-Segmented-Memory-Tag |

26	Delete-Packed-Object
27	Modify-Packed-Object
28	Write-Segments-6TypeD-Tag
29	Read-Segments-6TypeD-Tag
30	Write-Monomorphic-Ull
31	Configure-Extended-DSFID
32	Configure-Multiple-Records-Header
33	Read-Multiple-Records
34	Delete-Multiple-Record

Additional command and response modules, and their final arc values, will be added in numeric sequence, as required, to this part of ISO/IEC 15961.

The complete module specifies the function that the interrogator shall perform. Each command specifies, as appropriate, particular processes to be undertaken by the Data Processor, the Tag Driver, and the interrogator in communications across the air interface. Each response specifies, as appropriate, particular processes to be undertaken by the Tag Driver, the Data Processor, and the communications across the application interface.

9.2 Completion-Code

The **Completion-Code** is part of the response to each command. The **Completion-Code** is an INTEGER value that reports specifically on how the particular command was processed and executed, successfully or not. It is returned in each response. If its value is 0 (00₁₆), the command has been successfully executed. If its value is 255 (FF₁₆), the command could not be executed by the system for the reason specified in the **Execution-Code** (see 9.3). If its value is different from 0 and 255, it indicates that the command was not executed as instructed by the application for the reason mentioned.

NOTE The **Completion-Code** provides information on the basis that the command can be invoked for the particular RFID tag in the communication chain, whereas the **Execution-Code** indicates a systems error or success.

The following **Completion-Codes** apply:

- 0 **No-Error:** The command was successfully executed.
- 1 **AFI-Not-Configured:** The command could not be completed, possibly because of a prior configure action.
- 2 **AFI-Not-Configured-Locked:** The **AFI** was found to be locked (by a previous command execution), so this **AFI** could not be configured, as requested by the command, on this occasion.
- 3 **AFI-Configured-Lock-Failed:** The **AFI** was correctly configured and encoded, but the lock function could not be completed.
- 4 **DSFID-Not-Configured:** The command could not be completed, possibly because of a prior configure action.
- 5 **DSFID-Not-Configured-Locked:** The **DSFID** was found to be locked (by a previous command execution), so this **DSFID** could not be configured, as requested by the command, on this occasion.
- 6 **DSFID-Configured-Lock-Failed:** The **DSFID** was correctly configured and encoded, but the lock function could not be completed.
- 7 **Object-Locked-Could-Not-Modify:** The existing encodation on the RFID tag is locked, and as a result this attempt to update the **Object** value could not be completed.
- 8 **Singulation-Id-Not-Found:** The particular RFID tag, specified by the **Singulation-Id**, could not be found in the operating area.

- 9 **Object-Not-Added:** The data set of Precursor, **Object** and **Object-Identifier** could not be added to the Logical Memory Map (e.g. because there was insufficient memory space). This response also applies if the RFID tag supports a lock function that failed.
- 10 **Duplicate-Object:** An **Object-Identifier** with the same value as the one to be processed (added, modified, or deleted) was found. The process was aborted.
- 11 **Object-Added-But-Not-Locked:** The data set was added to a RFID tag that did not support a lock feature in the memory.
- 12 **Object-Not-Deleted:** The data set of Precursor, **Object-Identifier** and **Object** could not be deleted from the Logical Memory Map (e.g. because the delete function is not supported by the particular RFID tag).
- 13 **Object-Identifier-Not-Found:** The intended process could not be executed because the defined **Object-Identifier** is not actually encoded on the RFID tag.
- 14 **Object-Locked-Could-Not-Delete:** The **Object** is already locked and so cannot be deleted.
- 15 **Object-Not-Read:** The intention to read the specified **Object** failed.
- 16 **Objects-Not-Read:** The intention to read one or more of the specified **Objects** failed.
- 17 **Blocks-Locked:** The intended action to erase encoded bytes from one or more blocks could not be achieved because blocks were previously locked.
- 18 **Erase-Incomplete:** The intended action to erase encoded bytes from one or more blocks was interrupted and that not all blocks have been processed. The command can be re-invoked to complete the process.
- 19 **Read-Incomplete:** The intention to read the complete contents of the Logical Memory Map failed, because not all blocks from the RFID tag were transferred to the Logical Memory.
- 20 **System-Info-Not-Read:** The intention to read the system information failed.
- 21 **Object-Not-Modified:** The data set of Precursor, **Object-Identifier** and **Object** could not be modified on the Logical Memory Map (e.g. because the modify function is not supported by the particular RFID tag).
- 22 **Object-Modified-But-Not-Locked:** The **Object** was modified in a RFID tag that did not support a lock feature in the memory.
- 23 **Failed-To-Read-Minimum-Number-Of-Tags:** The minimum number of RFID tags were not identified with the specified selection criterion, possibly because of a time-out.
- 24 **Failed-To-Read-Exact-Number-Of-Tags:** The exact number of RFID tags were not identified with the specified selection criterion, possibly because of a time-out.
- 25 **Password-Mismatch:** The **Password** presented in the command did not match the equivalent password on the RFID tag, so the associated action was aborted.
- 26 **Password-Not-Written:** The **Password** was not written, either because the memory location was already encoded, or the feature was not supported.
- 27 **Zero-Kill-Password-Error:** The Kill operation was not executed because the **Kill-Password** has a zero value.
- 28 **Kill-Failed:** The intention to render the tag failed (e.g. because of insufficient power).