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

**Part 1:
Application interface**

*Technologies de l'information — Protocole de données relatif à
l'identification par radiofréquence (RFID) pour la gestion d'objets —
Partie 1: Interface d'application*





COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2021

All rights reserved. Unless otherwise specified, or required in the context of its implementation, 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
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword	viii
Introduction	ix
1 Scope	1
2 Normative references	1
3 Terms, definitions and abbreviated terms	1
3.1 Terms and definitions	1
3.2 Abbreviated terms and designations	2
4 Conformance	2
4.1 General	2
4.2 Application conformance	2
4.3 Conformance of the Data Processor	3
5 Protocol model	3
6 Presentation conventions	3
6.1 Commands, responses, and arguments	3
6.1.1 General	3
6.1.2 Data types	3
6.2 Object Identifier presentation in the application interface	4
6.2.1 Object identifier structure to ISO/IEC 8824-1	4
6.2.2 Presenting the Object-Identifier in accordance with ISO/IEC 8824-1	5
6.2.3 Presenting the Object-Identifier as a Uniform Resource Name (URN)	5
6.3 Byte notation	5
6.3.1 Byte — Basic unit for 8-bit coding	5
6.3.2 Bit ordering	5
6.3.3 Byte conversion	6
7 Processing application commands and responses	6
7.1 General	6
7.1.1 Option A: Straight through process	6
7.1.2 Option B: Transfer encoding	6
7.2 Encoding system related information in commands	7
7.2.1 Singulation-Id	7
7.2.2 AFI	7
7.2.3 DSFID	7
7.2.4 Access-Method	8
7.2.5 Data-Format	9
7.3 Preparing the basic Objects and other application-based arguments	10
7.3.1 General	10
7.3.2 General model	11
7.3.3 Object-Identifier	11
7.3.4 Relating Object-Identifiers	11
7.3.5 Object	11
7.3.6 Compact-Parameter	12
7.3.7 Object-Lock	14
7.4 Other command arguments	14
7.4.1 Access-Password	14
7.4.2 Additional-App-Bits	14
7.4.3 AFI-Lock	14
7.4.4 Append-To-Existing-Multiple-Record	14
7.4.5 Application-Defined-Record-Capacity	15
7.4.6 Avoid-Duplicate	15
7.4.7 Battery-Assist-Indicator	15
7.4.8 Block-Align	15
7.4.9 Block-Align-Packed-Object	15

7.4.10	Check-Duplicate	15
7.4.11	Data-CRC-Indicator	15
7.4.12	Data-Length-Of-Record	16
7.4.13	Delete-MR-Method	16
7.4.14	Directory-Length-EBV8-Indicator	16
7.4.15	DSFID-Lock	16
7.4.16	DSFID-Pad-Bytes	16
7.4.17	Editable-Pointer-Size	16
7.4.18	Encoded-Memory-Capacity	17
7.4.19	EPC-Code	17
7.4.20	Full-Function-Sensor-Indicator	17
7.4.21	Hierarchical-Identifier-Arc	17
7.4.22	Identifier-Of-My-Parent	17
7.4.23	Identify-Method	17
7.4.24	ID-Type	17
7.4.25	Instance-Of-Arc	18
7.4.26	Kill-Password	18
7.4.27	Length-Of-Mask	18
7.4.28	Lock-Directory-Entry	18
7.4.29	Lock-Multiple-Records-Header	18
7.4.30	Lock-Record-Preamble	18
7.4.31	Lock-UII-Segment-Arguments	18
7.4.32	Max-App-Length	18
7.4.33	Memory-Bank	19
7.4.34	Memory-Bank-Lock	19
7.4.35	Memory-Length-Encoding	19
7.4.36	Memory-Segment	19
7.4.37	Memory-Type	19
7.4.38	Multiple-Records-Directory-Length	19
7.4.39	Multiple-Records-Features-Indicator	19
7.4.40	NSI-Bits	20
7.4.41	Number-In-Data-Element-List	20
7.4.42	Number-Of-Records	20
7.4.43	Number-Of-Tags	20
7.4.44	Objects-Offsets-Multiplier	20
7.4.45	Packed-Object-Directory-Type	20
7.4.46	Password	21
7.4.47	Password-Type	21
7.4.48	PO-Directory-Size	21
7.4.49	PO-Index-Length	21
7.4.50	Pointer	21
7.4.51	Pointer-To-Multiple-Records-Directory	21
7.4.52	Read-Record-Type	21
7.4.53	Read-Type	23
7.4.54	Record-Memory-Capacity	23
7.4.55	Record-Type-Arc	23
7.4.56	Record-Type-Classification	24
7.4.57	Sector-Identifier	24
7.4.58	Simple-Sensor-Indicator	24
7.4.59	Start-Address-Of-Record	24
7.4.60	Tag-Data-Profile-ID-Table	24
7.4.61	Tag-Mask	25
7.4.62	Update-Multiple-Records-Directory	25
7.4.63	Word-Count	25
7.4.64	Word-Pointer	25
7.5	Command-related field names	25
7.5.1	General	25
7.5.2	Data-Set	25

7.5.3	Identities.....	25
7.5.4	Length-Lock Byte.....	25
7.5.5	Length-Of-Encoded-Data.....	26
7.5.6	Lock-Status.....	26
7.5.7	Logical-Memory-Map.....	26
7.5.8	Memory-Capacity.....	26
7.5.9	Module-OID.....	26
7.5.10	Number-Of-Tags-Found.....	26
7.5.11	PO-ID-Table.....	26
7.5.12	Protocol-Control-Word.....	26
7.5.13	Read-Data.....	26
7.6	Data security.....	26
8	Dataflows and processes to the air interface.....	27
8.1	General.....	27
8.2	Establishing communications between the application and the tag.....	27
8.2.1	General.....	27
8.2.2	Air interface services.....	27
8.2.3	System information.....	27
8.3	Application system services.....	28
9	Command-Codes, Completion-Codes, and Execution-Codes.....	28
9.1	General.....	28
9.2	Final arc values of the command and response modules.....	28
9.3	Completion-Code.....	30
9.4	Execution-Code.....	30
10	Commands and responses.....	30
10.1	General.....	30
10.2	Configure-AFI.....	30
10.2.1	Configure-AFI command.....	30
10.2.2	Configure-AFI response.....	31
10.3	Configure-DSFID.....	32
10.3.1	General.....	32
10.3.2	Configure-DSFID command.....	32
10.3.3	Configure-DSFID response.....	32
10.4	Inventory-Tags.....	33
10.4.1	Inventory-Tags command.....	33
10.4.2	Inventory-Tags response.....	34
10.5	Delete-Object.....	35
10.5.1	Delete-Object command.....	35
10.5.2	Delete-Object response.....	36
10.6	Modify-Object.....	36
10.6.1	Modify-Object command.....	36
10.6.2	Modify-Object response.....	37
10.7	Read-Object-Identifiers.....	38
10.7.1	Read-Object-Identifiers command.....	38
10.7.2	Read-Object-Identifiers response.....	38
10.8	Read-Logical-Memory-Map.....	39
10.8.1	Read-Logical-Memory-Map command.....	39
10.8.2	Read-Logical-Memory-Map response.....	39
10.9	Erase-Memory.....	40
10.9.1	Erase-Memory command.....	40
10.9.2	Erase-Memory response.....	40
10.10	Get-App-Based-System-Info.....	41
10.10.1	Get-App-Based-System-Info command.....	41
10.10.2		
	Get-App-Based-System-Info response.....	41
10.11	Write-Objects.....	41
10.11.1	Write-Objects command.....	41

10.11.2	Write-Objects response	43
10.12	Read-Objects	44
10.12.1	Read-Objects command	44
10.12.2	Read-Objects response	45
10.13	Write-Objects-Segmented-Memory-Tag	46
10.13.1	Write-Objects-Segmented-Memory-Tag command	46
10.13.2	Write-Objects-Segmented-Memory-Tag response	47
10.14	Write-EPC-UII	48
10.14.1	Write-EPC-UII command	48
10.14.2	Write-EPC-UII response	49
10.15	Inventory-ISO-UIImemory	49
10.15.1	Inventory-ISO-UIImemory command	49
10.15.2	Inventory-ISO-UIImemory response	50
10.16	Inventory-EPC-UIImemory	50
10.16.1	Inventory-EPC-UIImemory command	50
10.16.2	Inventory-EPC-UIImemory response	51
10.17	Write-Password-Segmented-Memory-Tag	51
10.17.1	Write-Password-Segmented-Memory-Tag command	51
10.17.2	Write-Password-Segmented-Memory-Tag response	51
10.18	Read-Words-Segmented-Memory-Tag	52
10.18.1	Read-Words-Segmented-Memory-Tag command	52
10.18.2	Read-Words-Segmented-Memory-Tag response	52
10.19	Kill-Segmented-Memory-Tag	53
10.19.1	Kill-Segmented-Memory-Tag command	53
10.19.2	Kill-Segmented-Memory-Tag response	53
10.20	Delete-Packed-Object	54
10.20.1	Delete-Packed-Object command	54
10.20.2	Delete-Packed-Object response	54
10.21	Modify-Packed-Object-Structure	55
10.21.1	Modify-Packed-Object-Structure command	55
10.21.2	Modify-Packed-Object-Structure response	56
10.22	Write-Segments-6TypeD-Tag	56
10.22.1	Write-Segments-6TypeD-Tag command	56
10.22.2	Write-Segments-6TypeD-Tag response	58
10.23	Read-Segments-6TypeD-Tag	59
10.23.1	Read-Segments-6TypeD-Tag command	59
10.23.2	Read-Segments-6TypeD-Tag response	60
10.24	Write-Monomorphic-UII	61
10.24.1	Write-Monomorphic-UII command	61
10.24.2	Write-Monomorphic-UII response	63
10.25	Configure-Extended-DSFID	63

10.25.1	General	63
10.25.2	Configure-Extended-DSFID command	64
10.25.3	Configure-Extended-DSFID response	64
10.26	Configure-Multiple-Records-Header	65
10.26.1	General	65
10.26.2	Configure-Multiple-Records-Header command	65
10.26.3	Configure-Multiple-Records-Header response	67
10.27	Read-Multiple-Records	68
10.27.1	Read-Multiple-Records command	68
10.27.2	Read-Multiple-Records response	69
10.28	Delete-Multiple-Record	69
10.28.1	Delete-Multiple-Record command	69
10.28.2	Delete-Multiple-Record response	70
11	Arguments	71
11.1	Add-Objects	71
11.2	DSFID-Constructs	71
11.3	EPC-UIIMemory	72
11.4	Ext-DSFID-Constructs	72
11.5	ISO-UIIMemory	73
11.6	Item-Related-Add-Objects	74
11.7	Item-Related-DSFID-Constructs	74
11.8	Multiple-Records-Constructs	74
11.9	Multiple-Records-Directory-Structure	75
11.10	Multiple-Records-Header-Structure	76
11.11	Multiple-Records-Preamble-Structure	77
11.12	Packed-Object-Constructs	78
11.13	Read-Objects	79
11.14	Read-Objects-Response	79
11.15	Read-UIDs-Response	80
11.16	UII-Add-Objects	80
11.17	UII-DSFID-Constructs	80
11.18	Write-Responses	80
Annex A	(informative) Abstract syntax and transfer encoding rules of ISO/IEC 15961:2004	82
Annex B	(informative) Accommodating established data formats	92
Annex C	(informative) Relating data Objects	94
Annex D	(informative) Data security issues	96
Annex E	(informative) Original commands and responses using ASN.1 abstract syntax	98
Annex F	(informative) Example of a transfer encoding to ISO/IEC 15961:2004	135
Bibliography		139

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.

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

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

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

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

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

This second edition cancels and replaces the first edition (ISO/IEC 15961-1:2013), which has been technically revised.

The main changes are as follows:

- Clauses from ISO/IEC 15962 which had been reproduced in this document have been removed and replaced by references to these clauses.
- The requirement for backwards compatibility with the 2004 version of this document (ISO/IEC 15961:2004) has been clarified.
- The document was edited for clarity and conformity with ISO/IEC Directives Part 2 drafting rules.

A list of all parts in the ISO/IEC 15961 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

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 tag is invisible and accessible between the tag and the interrogator only using 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 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 tag in one system's implementation and to be read at a later time in a completely different and unknown system's implementation. The data bits stored on each tag must be formatted in such a way as to be reliably read at the point of use if the tag is to fulfil its basic objective. This reliability is achieved through the specification of a Data Protocol in this document 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 document.

Manufacturers of RFID equipment (e.g. interrogators, tags) and users of RFID technology require standards-based Data Protocols for RFID for item management. This document, ISO/IEC 15962, ISO/IEC 24791-1, ISO/IEC 24791-2 and ISO/IEC 24791-5 specify these protocols, which are layered above the air interface standards defined in the ISO/IEC 18000 series.

The transfer of data to and from an application, supported by appropriate application commands, is the subject of this document. This document is intended to be used as a reference to develop software appropriate for RFID applications and equipment. ISO/IEC 15962, which is intended to be used with this document, specifies the overall process and the methodologies developed to format the application data into a structure to store on the tag.

NOTE ISO/IEC 15961:2004 is a withdrawn standard, replaced by ISO/IEC 15961-1, ISO/IEC 15961-2, ISO/IEC 15961-3 and ISO/IEC 15961-4. ISO/IEC 15961:2004 is referenced to point out the differences with this document because some systems still use the withdrawn version. All information pertaining to the use of the withdrawn ISO/IEC 15961:2004 is contained in this document. The intention is to remove reference to the withdrawn standard in the next version of this document.

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

Part 1: Application interface

1 Scope

This document 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 ISO/IEC 18000 series air interfaces.

This document:

- provides guidelines on presenting data 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 tag and the application;
- does not specify any required transfer syntax with ISO/IEC 15962, but provides the non-normative information in [Annex A](#) for backward compatibility with the 2004 version of this document (ISO/IEC 15961:2004¹⁾).

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements 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 15961-3, *Information technology — Data protocol for radio frequency identification (RFID) for item management — Part 3: RFID data constructs*

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

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

3 Terms, definitions and abbreviated terms

3.1 Terms and definitions

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

1) Withdrawn standard. Replaced by ISO/IEC 15961-1, ISO/IEC 15961-2, ISO/IEC 15961-3 and ISO/IEC 15961-4.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

- ISO Online browsing platform: available at <https://www.iso.org/obp>
- IEC Electropedia: available at <https://www.electropedia.org/>

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 1 to entry: This was called "data protocol processor" in ISO/IEC 15961:2004²⁾.

3.2 Abbreviated terms and designations

URN uniform resource name

Since "TDS", "Type C" and "Type D" are commonly used in the industry to refer to the technology components as specified by these standards, the following designations are used in this document:

Type C ISO/IEC 18000-63

Type D ISO/IEC 18000-64

TDS GS1 EPC Tag Data Standard

4 Conformance

4.1 General

The commands and responses in this document are expressed in an abstract syntax; transfer encoding, as specified by ISO/IEC 15961 (the superseded 2004 version of this document) is no longer required. As such, conformance to this document is specifically indicated by the resultant proper encoding of tags according to ISO/IEC 15962.

The arguments and fields contained in individual commands and responses identify what needs to be considered 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 a tag. Because of the way the Data Protocol is structured, the commands and responses specified in this document are, to a large extent, independent of particular tag types that are only known to the Data Processor through the Tag Driver. The result is that ISO/IEC 15962 can specify conformance requirements for valid encoding, which this document cannot.

This clause provides conformance best practice advice to achieve an integrated data communication channel between the application and the tag.

4.2 Application conformance

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 shall be considered in transfers between the application and the Data Processor.

2) Withdrawn standard. Replaced by ISO/IEC 15961-1, ISO/IEC 15961-2, ISO/IEC 15961-3 and ISO/IEC 15961-4

In particular, application standards shall 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 tag, and the necessary processes that the Data Processor shall 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 document 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

The ISO/IEC 15962 protocol model applies for this document.

6 Presentation conventions

6.1 Commands, responses, and arguments

6.1.1 General

Commands and responses are defined within a box format, such as is shown in [Figure 1](#).

Each command or response contains an ordered list of fields or arguments. 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.



Figure 1 — Box format for commands, responses and complex arguments

6.1.2 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.

- HEXADECIMAL ADDRESS: A location (on the memory of a tag), expressed as a hexadecimal value.
- INTEGER: An integer can take any whole number. In the context of this document, 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 document uses the OBJECT IDENTIFIER type as defined in ISO/IEC 8824-1 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 (see [Figure 2](#)). 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.

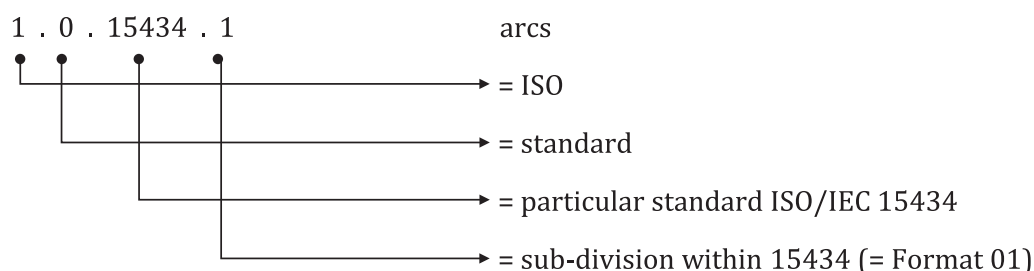


Figure 2 — Example object identifier arcs

The only top arcs permitted for all Object Identifiers are shown in [Table 1](#).

Table 1 — Object Identifier top arcs

Identifier arc name	Numerical value
itu-t	0
iso	1
joint-iso-itu-t	2

NOTE 1 Any ISO/IEC standard, e.g. this document, has 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 ISO/IEC 15961-3.

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 three arcs.

NOTE 2 This structure enables Object Identifiers from different domains (e.g. open and closed systems) to be encoded unambiguously on a 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 document.
- 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 tag. This structure is particularly important in applications that require a variety of data from a common data dictionary to be encoded on a tag. The Root-OID is either explicitly encoded or declared according to the encoding rules of ISO/IEC 15962.
- 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 this document. These structures are applied in situations where a common root applies to the set of Object Identifiers to be encoded on the tag. For example, if all the Object Identifiers have the common root 1 0 15961 12 encoding spaces can be saved on the 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 RootOID, which is either encoded or declared in some other way.

NOTE 3 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.

6.2.2 Presenting the Object-Identifier in accordance with ISO/IEC 8824-1

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

1 0 15961 12 1

NOTE 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 Byte — Basic unit for 8-bit coding

This document supports binary, 6-bit, 7-bit, 8-bit, and user data that can 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.

An 8-bit byte is represented by two hexadecimal values, using the characters 0-9, and 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](#).

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 (ISO/IEC 15961:2004) 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 (e.g. 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 15961:2004 into a multi-part standard, this document now includes support for a "straight through" process.

There are two fundamentally different ways to implement support for the commands and responses defined in [Clause 10](#).

7.1.1 Option A: Straight through process

In devices or software that incorporate data input of commands compatible with this document 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 [7.3.3](#)) 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 document places no constraints on the process adopted.

7.1.2 Option B: Transfer encoding

If a device, or software only supports the processes of this document 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 a 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. It identifies the 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 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 tag, as specified in the ISO/IEC 18000 series.
- b) A data related identifier, e.g. like a Unique Item Identifier (UII), that provides for uniqueness within the specific 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 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 ISO/IEC 15961-3, as follows:

- If in the range 00_{16} to $0F_{16}$ the code applies to a closed system application, as defined in ISO/IEC 15961-3. 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 ISO/IEC 15961-3. 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 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 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 tag shall be 00_{16} .

7.2.3 DSFID

The DSFID (previously defined as StorageFormat in ISO/IEC 15961:2004) refers to specific identifiers that enable efficient encoding on tags. The value of the DSFID for RFID for Item Management is a single byte, or multiple bytes. 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 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 assigned by revision or amendment to this document.

If the Data-Format has a value between 32 and 287 as defined in ISO/IEC 15961-3, 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 document.

7.2.4 Access-Method

The Access-Method defines the way data can be mapped on the tag and be accessed from the tag. The value of Access-Method should be stored on the 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 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.

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

- 2 Packed-Objects: The Packed-Object structure was introduced at the first revision of ISO/IEC 15962 and this edition of this document. 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 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 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 several 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 the same Object-Identifiers to be encoded on all 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 conform 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 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 can 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 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 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 must 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, considering 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 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 tag. The Data-Format enables data Objects to be identified 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 than that defined by the Data-Format) can still be encoded on the 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 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 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 ISO/IEC 15961-3. The following are relevant to this document:

- | | |
|---|---|
| 0 | Not-Formatted: This is used for tags not formatted, or not yet formatted, to this document. |
| 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 tag. For example, it can 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 tag has a common RootOID, which does not conform with one of the specific Root-OIDs associated with a Data-Format registered under the rules of ISO/IEC 15961-2. |

The Root-OID for Data-Format 2 shall be directly encoded on the tag using the appropriate root arcs. Each Object is encoded on the tag using a Relative-OID, representing the remaining lower order arcs.

- | | |
|-----------|---|
| 3 to 28 | These code values apply to an open system application as registered under ISO/IEC 15961-2. 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 in accordance with this document and ISO/IEC 15962. |
| 30 | This code applies to closed system data, where the encoding rules are not in accordance 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 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 ISO/IEC 15961-2, 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

7.3.1 General

This is an initial process to ensure that the data Objects are prepared in a format compatible with this document.

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

7.3.2 General model

[Figure 3](#) 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 document, has a data dictionary, or list of data Objects or data elements.

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

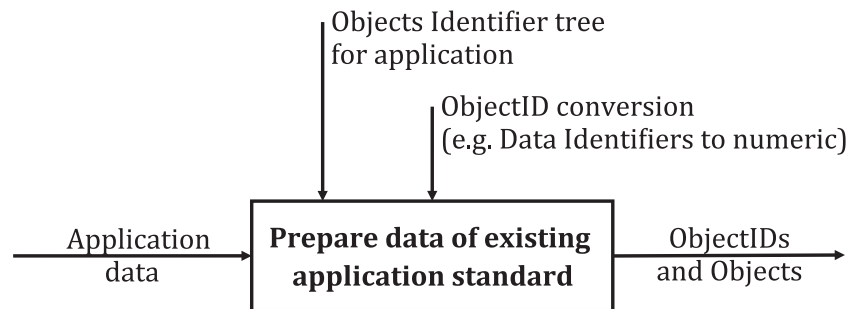


Figure 3 — Data flow model: Prepare basic Objects

7.3.3 Object-Identifier

The Object-Identifier shall be provided by the application system as a series of arcs as defined in [6.2](#) 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 tag, based on the Data-Format received from the tag.

7.3.4 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.

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

7.3.5 Object

7.3.5.1 General

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 interpretation is given in the following subclauses.

7.3.5.2 Basic 8-bit character set information

Many business applications make use of ISO/IEC 8859-1 or 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 shall 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.

7.3.5.3 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.

7.3.5.4 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 can 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](#).

7.3.6 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; 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 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, using 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.

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 ISO/IEC 15961-2.

- 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.

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 ISO/IEC 15961-2.

- 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.

Compaction is generally applied because it increases the encodation efficiency on the 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 will 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 can 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 can 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 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 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 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 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.

7.3.7 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 processed in this way, permanently encoded on the tag, or only capable of being unlocked in some types of 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 tag. A match results in additional actions on the 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 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 tag can only be used in the way prescribed. The locking process shall have the effect of making the bytes processed in this way, permanently encoded on the tag, or only capable of being unlocked in some types of 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 AFI-Lock does not apply.

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 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 can 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 [10.28.1](#).

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 can originally be small in size, however, it 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 tag can only be used as prescribed. The locking process shall have the effect of making the bytes processed in this way permanently encoded on the tag, or only capable of being unlocked in some types of 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 can 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 can 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 TDS. 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 shall be rounded to 8-bit bytes for processing through the Data Processor and be rounded to 16-bit words for encoding on the associated tag.

7.4.20 Full-Function-Sensor-Indicator

The command argument Full-Function-Sensor-Indicator is used by the application that the 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 tags belonging to the 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 tags to be read using the Number-Of-Tags (see [7.4.43](#)). More precise advice is provided in Inventory-Tags command (see [10.4](#)).

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 tag. A match results in the 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 TDS data encoded in the UII memory of a segmented memory 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 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 | Number-of-records field remains unlocked, the preceding fields are locked |
| 3 | Data Length of the Directory field remains unlocked, all other fields including the number-of-records field are locked |
| 4 | 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 a 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 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 where on a segmented memory tag data is to be encoded. It is similar in function to the Memory-Bank argument (see 7.4.33), except that the tag for which this argument is appropriate can address multiple segments in the same air interface transaction.

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 1_2 indicates that all the records use the same Access-Method. If this bit = 0_2 then bits 7 to 4 = 0000_2
- Bits 7 to 4 identify the Access-Method if this is applied consistently. If bit 8 = 0_2 , the currently permissible string for bits 7 to 4 are:

0000_2	indicates that all records use the No-Directory Access-Method
0001_2	is not permitted
0010_2	indicates that all records use the Packed-Objects Access-Method
0011_2	indicates that all records use the Tag-Data-Profile Access-Method
0100_2	is not permitted

0101₂ to 1111₂ is currently reserved for additional Access-Methods

- Bit 3 with the value 1₂ 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 0₂ indicates that the number-of-records field can be incorrect.
- Bit 1 with the value 1₂ indicates that there is an additional byte for indicating additional features. Currently this is RFU and therefore set to 0₂.

7.4.40 NSI-Bits

The command argument NSI-Bits defines a 9-bit code defined by TDS used as a prefix to the EPC-Code when encoded on the associated 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:

- 0₂ 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.
- 1₂ 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 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 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 by setting a 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 tag. A match provides permissions to carry out additional actions with the 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 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 TDS data encoded in the UII memory of a segmented memory 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 of 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 may 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 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 numbers.

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-OIDs-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-OIDs-Response-List, i.e. either with the set of instance-of arcs or with the set of hierarchical id arcs.

If Read-OIDs-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-OIDs-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 shall 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 structured, and therefore differs from the superseded 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 override 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 binary bit string that identifies the class of record being encoded. The following codes apply:

000 ₂	stand-alone record, with an instance-of arc = 0
001 ₂	stand-alone record, with an instance-of arc >0
010 ₂	hierarchical record, top level
011 ₂	hierarchical record, has both parent and child(ren)
100 ₂	hierarchical record, data element list
101 ₂	other hierarchical record, no further children (never applies to a data element list)
110 ₂	Not relevant to this command (because it is associated with deleted records)
111 ₂	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.

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, shall 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 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 the 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 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 TDS data encoded in the UII memory of a segmented memory 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 consider 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 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 tag from which an encoded byte string is to be read.

7.5 Command-related field names

7.5.1 General

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

7.5.2 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 a tag.

7.5.3 Identities

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

7.5.4 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 provides information about the lock status of pages on the tag.

7.5.5 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.

7.5.6 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.

7.5.7 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 tag.

7.5.8 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.

7.5.9 Module-OID

The field Module-OID identifies an individual command or response module with a full Object-Identifier as defined in [7.3.3](#).

7.5.10 Number-Of-Tags-Found

The field name Number-Of-Tags-Found is a command-response field that returns the actual number of 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.

7.5.11 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.

7.5.12 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 tag. The Protocol-Control-Word contains bit-based data that identifies other features supported on the tag.

7.5.13 Read-Data

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

7.6 Data security

The data (Object) may be made secure using 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 document and ISO/IEC 15962.

NOTE Additional guidance is provided in [Annex D](#).

8 Dataflows and processes to the air interface

8.1 General

Various processes are required to, e.g. format the tag, to write data to it, to read from it, to modify data. These are defined in the subclauses that follow. All of the processes to write and add data are described. Where the read process is the inverse of the write process, this is described briefly, otherwise a further description is provided.

8.2 Establishing communications between the application and the tag

8.2.1 General

The Data Processor does not communicate directly with the tag, but through the Tag Driver. The Data Processor requires specific system information based on the configuration of the tag (see [8.2.3](#)). This is to set parameters of the Logical Memory to correctly represent the 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 [8.2.2](#)).

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

8.2.2 Air interface services

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

- Application memory is an integer number of bytes.

NOTE The term application memory is used in this subclause as a generic name for the area of tag memory available for user data (this is sometimes called "user memory" in the ISO/IEC 18000 series) 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.
- 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 tag shall have a mechanism for storing the system information (see [8.2.3](#)), including the ability to write and read the component elements.

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

8.2.3 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 document 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](#)).

8.3 Application system services

The application system shall provide the following:

- Object Identifier (see [7.3.3](#));
- Object (see [7.3.5](#));
- Compact-Parameter (see [7.3.6](#));
- Object-Lock (see [7.3.7](#)).

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

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

9.1 General

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 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 document to the final arc of the command and response modules (see [9.2](#)). Completion-Codes (see [9.3](#)) and Execution-Codes (see [9.4](#)) 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 can be detected. The command shall be aborted with no data transferred to or from the tag in which the error is detected. This is possible because processing is done in the Logical Memory. Although other error conditions can 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 tags, all tags processed prior to the one with the detected error should be processed. The detection of the error aborts all subsequent processing.

The following subclauses define all the application commands and responses that are supported by this document. In addition to the basic syntax, these subclauses also describe the function and purpose of specific command arguments and responses. The commands and responses are grouped logically together. Within this subclause, the current appropriate abstract syntax is shown.

[Annex E](#) shows the original 16 modules using the ASN.1 abstract syntax and 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 is described in the relevant subclauses. [Annex F](#) shows an example of the original transfer encoding of a command and response.

9.2 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-Object-ID. The final arc shall be specific to the command/response pair and the following final arcs are as specified:

- 1 Configure-AFI
- 2 Configure-DSFID
- 3 Inventory-Tags
- 4 Reserved (used by withdrawn ISO/IEC 15961:2004)
- 5 Delete-Object
- 6 Modify-Object
- 7 Reserved (used by withdrawn ISO/IEC 15961:2004)
- 8 ReadObject-Identifiers
- 9 Reserved (used by withdrawn ISO/IEC 15961:2004)
- 10 Read-Logical-Memory-Map
- 11 Reserved (used by withdrawn ISO/IEC 15961:2004)
- 12 Erase-Memory
- 13 Get-App-Based-System-Info
- 14 Reserved (used by withdrawn ISO/IEC 15961:2004)
- 15 Reserved (used by withdrawn ISO/IEC 15961:2004)
- 16 Reserved (used by withdrawn ISO/IEC 15961:2004)
- 17 Write-Objects
- 18 Read-Objects
- 19 Write-Objects-Segmented-Memory-Tag
- 20 Write EPC-UII
- 21 Inventory-ISO-UIImemory
- 22 Inventory-EPC-UIImemory
- 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-UII
- 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 document.

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

9.3 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 command was processed and executed, successfully or not. It is returned in each response. If its value is 0 (00_{16}), the command has been successfully executed. If its value is 255 (FF_{16}), the command cannot be executed by the system for the reason specified in the Execution-Code (see 9.4). 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 tag in the communication chain, whereas the Execution-Code indicates a systems error or success.

The Completion-Codes as specified by ISO/IEC 15962 shall apply.

9.4 Execution-Code

The Execution-Code is part of the response to each command. The Execution-Code is an INTEGER value that reports on the way the command was processed and executed by the system, successfully or not. It is returned in each response. If its value is 0 (00_{16}), the command has been successfully processed, i.e. the protocol was executed. Other values indicate that a system error has occurred.

The Execution-Codes as specified by ISO/IEC 15962 shall apply.

10 Commands and responses

10.1 General

Each pair of command and response is specified in 10.2 to 10.22. Those arguments that require a separate specification are defined in 10.28.2.

10.2 Configure-AFI

10.2.1 Configure-AFI command

The AFI is a single byte code and is used as part of the selection process in an application. Details of the AFI codes assigned to applications are available on the Register of Data Constructs provided by the Registration Authority of ISO/IEC 15961-2:2019. If the air interface supports the AFI feature, only tags with a specific AFI will be returned for future processing.

This command is applicable for those air interface protocols where either of the following characteristics is true:

- A. The application command is directly related to an equivalent air interface command.
- B. The application command requires the AFI to be written to a particular memory location and nominally separated from item-related data by using a generic air interface write command.

The Configure-AFI command has the following arguments:

- AFI (see [7.2.2](#));
- AFI-Lock (see [7.4.3](#));
- Singulation-Id (see [7.2.1](#)).

The AFI-Lock argument applies to characteristic A above without constraint but can only apply to characteristic B above if the air interface protocol supports the locking of a single byte position defined for the AFI.

Configure-AFI command

Module-OID: OBJECT IDENTIFIER = 1 0 15961 126 1

Singulation-Id: BYTE STRING (0..255)

AFI: BYTE

Possible Values:

<u>Value</u>	<u>Definition</u>
00 ₁₆ – 0F ₁₆	As defined in ISO/IEC 15961-3
90 ₁₆ – CE ₁₆	As published by the Registration Authority of ISO/IEC 15961-2:2019
CF ₁₆	Reserved as an extension code

AFI-Lock: BOOLEAN

If set to TRUE, the interrogator shall lock the AFI

10.2.2 Configure-AFI response

The Configure-AFI response has the following field names:

- Completion-Code (see [9.3](#));
- Execution-Code (see [9.4](#)).

Configure-AFI response

Module-OID: OBJECT IDENTIFIER = 1 0 15961 127 1

Completion-Code: INTEGER

Possible Values:

<u>Value</u>	<u>Definition</u>
0	No-Error
1	AFI-Not-Configured
2	AFI-Not-Configured-Locked
3	AFI-Configured-Lock-Failed
8	Singulation-Id-Not-Found
255	Execution-Error

Execution-Code: INTEGER

Possible Values: As defined in [9.4](#).

10.3 Configure-DSFID

10.3.1 General

Configure-Extended-DSFID (see [10.25](#)) is required when encoding indicators of other features in the tag, such as the use of a Data-CRC. That command can also be better to encode an Access-Method with a value greater than 3, or a Data-Format with a value greater than 31.

10.3.2 Configure-DSFID command

The DSFID is a single-byte code that is used to reduce the encoding of Object-Identifiers and to define particular encoding rules for ISO/IEC 15962 to follow. These encoding rules apply to the Access-Method. Details of the Data-Format (part of the DSFID) assigned to applications are available on the Register of Data Constructs provided by the Registration Authority of ISO/IEC 15961-2:2019.

The command is applicable for those air interface protocols where either of the following characteristics is true:

- a) The application command is directly related to an equivalent air interface command.
- b) The application command requires the DSFID to be written to a particular location and memory nominally separated from item-related data but using a generic air interface write command.

The Configure-DSFID command has the following arguments:

- DSFID-Constructs (see [11.2](#));
- DSFID-Lock (see [7.4.15](#));
- Singulation-Id (see [7.2.1](#)).

The DSFID-Lock argument applies to characteristic A above without constraint but can only apply to characteristic B if the air interface protocol supports the locking of a single byte position defined by the DSFID.

Configure-DSFID command

Module-OID: OBJECT IDENTIFIER = 1 0 15961 126 2

Singulation-Id: BYTE STRING (0..255)

DSFID-Constructs-list: List of <DSFID-Constructs>

DSFID-Lock: BOOLEAN

If set to TRUE, the interrogator shall lock the DSFID

10.3.3 Configure-DSFID response

The Configure-DSFID response has the following field names:

- Completion-Code (see [9.3](#));
- Execution-Code (see [9.4](#)).

Configure-DSFID response

Module-OID: OBJECT IDENTIFIER = 1 0 15961 127 2

Completion-Code: INTEGER

Possible Values:

<u>Value</u>	<u>Definition</u>
0	No-Error

4	DSFID-Not-Configured
5	DSFID-Not-Configured-Locked
6	DSFID-Configured-Lock-Failed
8	Singulation-Id-Not-Found
255	Execution-Error

Execution-Code: INTEGER

Possible Values: As defined in [9.4](#).

10.4 Inventory-Tags

10.4.1 Inventory-Tags command

The Inventory-Tags command requires that the value of the AFI is specified to select tags belonging to a particular class, typically containing tags belonging to a defined domain. The Inventory-Tags command is intended to read a set of Singulation-Ids from tags that have a particular AFI. It is only applicable where an air interface command supports an inventory process using the AFI as a named argument and, generally, where a unique chip identifier is used in the arbitration process.

An additional selection criterion (Identify-Method) determines how many tags, conforming with the specified AFI selection criterion, need to be identified before the response can be provided. A mechanism that can be used to detect any tag entering the operating area, is to set the Inventory-At-Least field to 1. Certain conditions can be confirmed by only undertaking a partial inventory, i.e. by using either the Inventory-At-Least, or the Inventory-No-More-Than field. A reconciliation of a known quantity of previous transactions (e.g. to identify that all items intended to be in a container are actually there) can be achieved by using the Inventory-Exactly field.

The Inventory-Tags command has the following arguments:

- AFI (see [7.2.2](#));
- Identify-Method (see [7.4.23](#));
- Number-Of-Tags (see [7.4.43](#)).

Inventory-Tags command	
Module-OID: OBJECT IDENTIFIER = 1 0 15961 126 3	
AFI: BYTE	
Identify-Method: INTEGER (0..15)	
<i>Possible Values:</i>	
<u>Value</u>	<u>Definition</u>
0	Inventory-All-Tags
1	Inventory-At-Least
2	Inventory-No-More-Than
3	Inventory-Exactly
4 – 15	Reserved
Number-Of-Tags: INTEGER (0..65535)	

If the Identify-Method is set to Inventory-All-Tags, the interrogator shall perform a complete inventory of all tags present in its field of operation. The value of Number-Of-Tags is irrelevant and should be set to zero by the application.

If the Identify-Method is set to Inventory-At-Least, the interrogator shall perform an inventory of the tags present in its field of operation and (possibly) continue waiting until it has identified a number of tags equal to Number-Of-Tags. If the Number-Of-Tags is set to 1, the Interrogator shall wait until the

first tag has been detected. This is a mechanism to wait for a tag to enter the interrogator field. If the Number-Of-Tags is set to more than 1, the Interrogator waits until the specified number of tags has been detected.

If the Identify-Method is set to Inventory-No-More-Than, the interrogator shall initiate an inventory of the tags present in its field of operation and shall return a response with a number of tags lower or equal to Number-Of-Tags. The interrogator may interrupt the inventory process when the Number-Of-Tags has been reached or may continue the inventory process until all tags have been read.

NOTE 1 This can be constrained by the air interface and anti-collision mechanism.

If the Identify-Method is set to Inventory-Exactly, the interrogator shall initiate an inventory of the tags present in its field of operation and shall return a response with the number of tags equal to Number-Of-Tags. This command parameter can be used to confirm the actual number of tagged items in a container. The Interrogator will wait until the specified number of tags has been detected. The interrogator may interrupt the inventory process when the Number-Of-Tags has been reached or may continue the inventory process until all tags have been read.

NOTE 2 This can be constrained by the air interface and anti-collision mechanism.

Execution of this command with the arguments Inventory-At-Least and Inventory-Exactly can cause the interrogator to wait until sufficient tags enter its field of operation; also, the command-response cannot be initiated until after this delay. It is the responsibility of the application to accommodate this potentiality.

10.4.2 Inventory-Tags response

The Inventory-Tags response has the following field names:

- Completion-Code (see [9.3](#));
- Execution-Code (see [9.4](#));
- Identities (see [7.5.3](#));
- Number-Of-Tags-Found (see [7.5.10](#)).

Inventory-Tags response	
Module-OID: OBJECT IDENTIFIER = 1 0 15961 127 3	
Completion-Code: INTEGER	
<i>Possible Values:</i>	
<u>Value</u>	<u>Definition</u>
0	No-Error
23	Failed-To-Read-Minimum-Number-Of-Tags
24	Failed-To-Read-Exact-Number-Of-Tags
255	Execution-Error
Execution-Code: INTEGER	
<i>Possible Values:</i> As defined in 9.4 .	
Number-Of-Tags-Found: INTEGER (1..65535)	
Identities: List of <Singulation-Id>	

10.5 Delete-Object

10.5.1 Delete-Object command

The Delete-Object command instructs the interrogator to delete a defined Object-Identifier and its Object and associated parameters. Only one tag and only one Object-Identifier shall be programmed per command to ensure that the deletion process is robust. The delete function requires the removal of the Object-Identifier, the associated Object, precursor and other components of the data set from the Logical Memory Map, and then the re-writing of any data sets at higher address locations on the tag.

For the No-Directory and Directory Access-Methods, if other encoding follows the deleted Object-Identifier, the Data Processor may replace the deleted bytes with a null Data-Set. This procedure is invoked automatically by the Data Processor.

For the Packed-Objects Access-Method, the interrogator shall delete the Object from within the first Packed-Object in which the Object-Identifier exists. If the Packed-Object is editable, the Packed-Objects editing procedures shall be used so that the entire memory content does not need to be rewritten. Otherwise, the interrogator shall rewrite the content of the user memory appropriately recalculated with the Object-Identifier and its Object and associated parameters deleted. If any of the memory that is required to be rewritten during the execution of this command is locked, the response will return the appropriate Completion-Code.

If the Access-Method is Tag-Data-Profile, deletion is not possible because of the fixed structure of the encoding and the fact that there are no system-based null characters, and the appropriate completion code shall be returned.

For Multiple-Records the Object-Identifier shall identify a data element within an individual record. The rules for the inherent Access-Method shall apply. To delete the entire record, see [10.28](#).

The Delete-Object command instructs the interrogator to delete the data set specified by its Object-Identifier from the tag Logical Memory Map. There is a possibility that this procedure will not succeed if the data set is locked. If this is found to be the case, the response returns the appropriate Completion-Code. If the Check-Duplicate flag is set to TRUE, the interrogator shall verify, before deleting the requested Object, that there is only a single instance of the requested Object-Identifier. If the interrogator detects that the tag is encoding more than one instance of the referenced Object-Identifier, it shall not perform the Delete-Object function and shall return the appropriate Completion-Code.

If the Check-Duplicate flag is set to FALSE, the interrogator shall delete the first occurrence of the data set specified by its Object-Identifier.

This is an argument that effectively provides no protection against duplicate Object-Identifiers. It should only be used when there is a high expectation of no duplicates.

The Delete-Object command has the following arguments:

- Check-Duplicate (see [7.4.10](#));
- Object-Identifier (see [7.3.3](#));
- Singulation-Id (see [7.2.1](#)).

Delete Object command

Module OID: OBJECT IDENTIFIER = 1 0 15961 126 5

Singulation Id: BYTE STRING (0..255)

Object Identifier: OBJECT IDENTIFIER

Check Duplicate: BOOLEAN

If set is TRUE, the interrogator shall check that there is only one occurrence of the Identifier Object

10.5.2 Delete-Object response

The Delete-Object response has the following field names:

- Completion-Code (see [9.3](#));
- Execution-Code (see [9.4](#)).

Delete-Object response	
Module-OID: OBJECT IDENTIFIER = 1 0 15961 127 5	
Completion-Code: INTEGER	
<i>Possible Values:</i>	
<u>Value</u>	<u>Definition</u>
0	No-Error
8	Singulation-Id-Not-Found
10	Duplicate-Object
12	Object-Not-Deleted
13	Object-Identifier-Not-Found
14	Object-Locked-Could-Not-Delete
37	Data-CRC-Not-Applied
38	Length-Not-Encoded-In-DSFID
255	Execution-Error
Execution-Code: INTEGER	
<i>Possible Values:</i> As defined in 9.4 .	

10.6 Modify-Object

10.6.1 Modify-Object command

The Modify-Object command is intended to change the value of a data Object associated with an Object-Identifier already encoded on the memory of the tag. The complete memory shall be read to ensure that the Object-Identifier is not duplicated. If so, the command is aborted. Invoking this command depends on the Access-Method declared for the tag. In addition, the procedure is different if the resultant encoding length of the modified Object is different from the original length. Each of the cases is discussed under the appropriate Access-Method.

This command shall not be used to modify a Monomorphic-UII. If the AFI on the tag declares that it is registered for a Monomorphic-UII, the appropriate error shall be returned, and the encoding process aborted. The correct command to use is defined in [10.24](#).

The Modify-Object command instructs the interrogator to carry out three related processes:

- a) Read the complete Logical Memory Map from the tag.
- b) Identify the encoded packet (e.g. Data-Set, or Packed-Object, or Tag-Data-Profile) specified by the Object-Identifier. If duplicated instances are found, the process is aborted.
- c) Over-write with the modified encoded packet:
 - including re-structuring the Precursor for a Data-Set,
 - including any Packed-Object structuring rules,
 - including any pad bytes for shorter data in a Tag-Data-Profile.