INTERNATIONAL STANDARD

ISO/IEC 15961-1

First edition
2013-03-15

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

Part 1: Application interface

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

Partie 1: Interface d'application

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

Contents

Page

Foreword

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

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

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

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

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

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

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

— *Part 1: Application interface*

— *Part 2: Registration of RFID data constructs*

— *Part 3: RFID data constructs*

The following part is under preparation:

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

Introduction

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

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

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

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

Part 1: Application interface

1 Scope

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

This part of ISO/IEC 15961

— provides guidelines on how data shall be presented as objects;

— defines the structure of Object Identifiers, based on ISO/IEC 9834-1;

— specifies the commands that are supported for transferring data between an application and the radio frequency identification (RFID) tag;

— specifies the responses that are supported for transferring data between the RFID tag and the application;

— does not specify any required transfer syntax with ISO/IEC 15962, but provides the non-normative information in Annex A to provide backward compatibility with ISO/IEC 15961:2004.

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

2 Normative references

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

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

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

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

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

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

3 Terms, definitions and conventions

3.1 Terms and definitions

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

3.1.1
application
software component that issues commands and receives responses to the commands within a system

3.1.2
Data Processor
implementation of the processes defined in ISO/IEC 15962, including the Data Compactor, Formatter, Logical Memory, and Command/Response Unit

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

3.1.3
Relative-OID
particular Object Identifier where a common Root-OID (for the first and subsequent arcs) is implied, and remaining arcs after the Root-OID are defined by the Relative-OID

3.2 Conventions

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

4 Compliance

4.1 General

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

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

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

4.2 Application compliance

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

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

4.3 Conformance of the Data Processor

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

5 Protocol model

5.1 Overview

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

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

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

5.2 Layered protocol

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

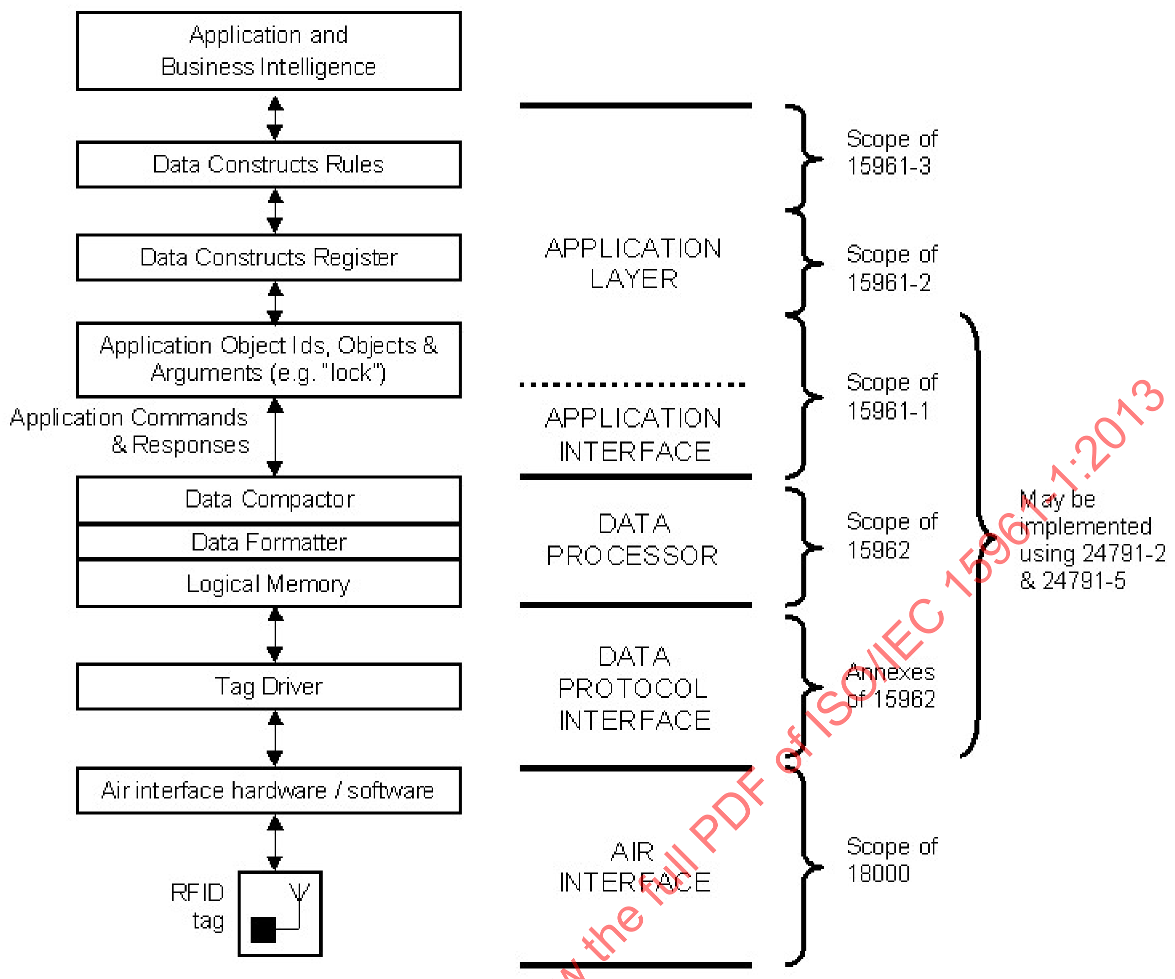


Figure 1 — Schematic of protocol layers for an implementation of RFID for item management

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

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

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

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

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

To illustrate how the functions and arguments are assembled into a structured format, a number of commands and responses have been constructed using an abstract syntax. This is independent of the host application, operating system, and program language and also independent of the specific command structures between the interrogator and tag driver. The abstract syntax used in this part of ISO/IEC 15961 is similar to that used in ISO/IEC 24791-5, and is intended to enable closer integration with that standard. The original version of ISO/IEC 15961:2004 included commands defined using ASN.1 abstract syntax. For backward compatibility the commands that were originally defined in this manner have been included in an annex in this part of ISO/IEC 15961.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

5.3 Flexible implementation configurations

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

— with ISO/IEC 15962 incorporated into the software system infrastructure architecture defined in ISO/IEC 24791. This is the recommended approach for any networked application.

— with this part of ISO/IEC 15961 and ISO/IEC 15962 incorporated into stand-alone software or devices that have as its output conformant encoding and / or conformant decoding with responses compliant to the responses of this part of ISO/IEC 15961.

5.4 Functional processes – interrogator implementation

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

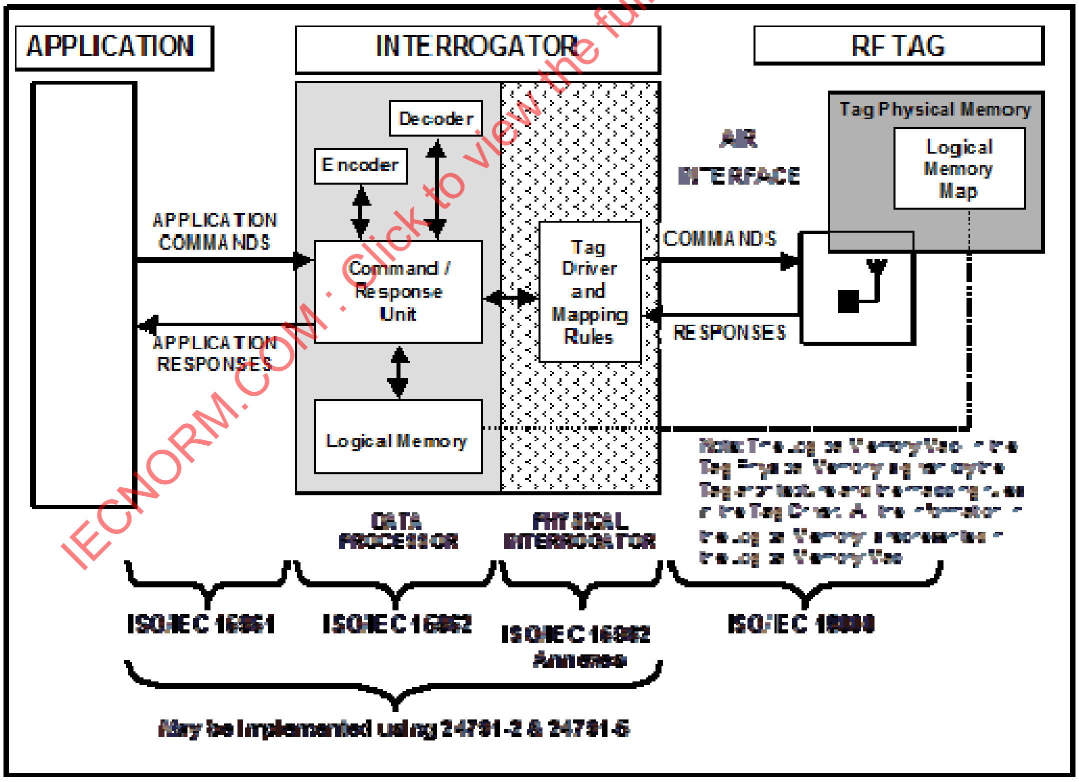


Figure 2 — Logical functions and interfaces of ISO/IEC 15962 with other RFID system components

5.4.1 Functional processes - application interface

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

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

5.4.2 Functional processes – interrogator

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

5.4.2.1 Data Processor

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

5.4.2.1.1 Command/Response unit

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

EXAMPLE

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

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

5.4.2.1.2 Logical Memory

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

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

5.4.2.1.3 Encoder

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

5.4.2.1.4 Data Compactor

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

5.4.2.1.5 Formatter

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

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

5.4.2.1.6 Decoder

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

5.4.2.2 Tag Driver

The Tag Driver provides two main functions:

— It provides mapping rules on the structure of the RFID tag to enable the contents of the Logical Memory in the Data Processor to be exchanged with the Logical Memory Map of the RFID tag in use.

— It provides facilities that accept the application commands of this data protocol, and converts them to a format that results in calls to command codes supported by the particular RFID Tag. For example, an application command **write Data Object {name}** could result in the RFID tag related command of write (block #, data).

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

5.4.2.3 Transfer mechanisms

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

5.4.3 RFID tag

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

5.5 ISO/IEC 15962 and the Data Processor

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

— Having clearly defined rules for compacting data that are applied independently of the application and the type of RFID tag.

— Defining fundamental structuring rules for the encoding of data largely determined by a user-selectable feature called the **Access-Method**.

— Having precise structuring rules within each **Access-Method**, including a clearly defined syntax that allows reading systems to selectively identify data required for the application.

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

6 Presentation conventions

6.1 Presentation of commands, responses and arguments

6.1.1 Commands and responses

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

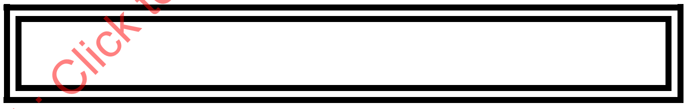


Figure 3 — Box format for commands and responses

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

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

6.1.2 Arguments

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

Figure 4 — Box format for arguments

6.1.3 Data types

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

— BOOLEAN: An argument that can have the values TRUE or FALSE.

— BIT STRING: A sequence of bits.

— BYTE: An integer with the possible values 0 to 255, usually expressed as a hexadecimal value 00_{16} to FF_{16}.

— BYTE STRING: A sequence of bytes. (Equivalent to OCTET STRING)

— EBV-8: A binary method to encode variable size numbers in the same field by using a leading indicator bit preceding a 7-bit value. The final EBV-8 component begins with a 0, all preceding components begin with a 1. For example, 69_{10} = **0**1000101, whereas 369_{10} = 101110001_2 whose EBV-8 representation is **1**0000010 **0**1110001 (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 an RFID tag), expressed as a hexadecimal value

— INTEGER: An integer can take any whole number. In the context of this part of ISO/IEC 15961, the values are all positive.

— OBJECT IDENTIFIER: An Object Identifier as defined in 6.2.1.

6.2 Object Identifier presentation in the application interface

6.2.1 Object identifier structure to ISO/IEC 8824-1

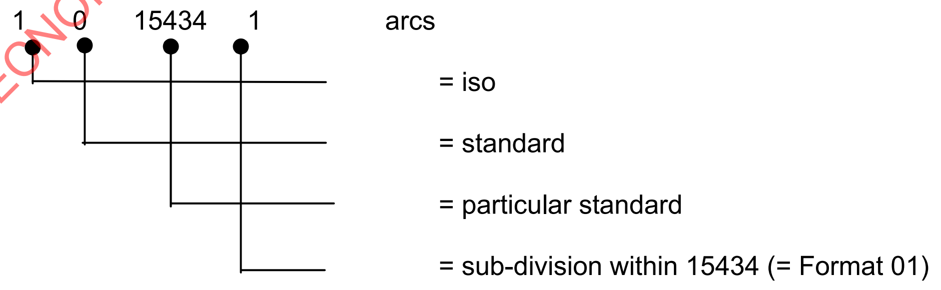
This part of ISO/IEC 15961 uses the OBJECT IDENTIFIER type as defined in ISO/IEC 8824-1 and with identifiers assigned as specified in ISO/IEC 9834-1. This uses a registration tree with a common implied root node (ISO/IEC 9834-1), a series of arcs from each node, with new arcs added as required to define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

— has a defined set of arcs to identify itself

— can manage the allocation of arcs under its node, independently of other bodies

— is assured of uniqueness from all other arcs in the registration tree

EXAMPLE



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

Table 1 — Object Identifier Top Arcs

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

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

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

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

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

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

— **Object-Identifier**: This full structure is used for communications between the application and the Data Processor defined by the scope of this part of ISO/IEC 15961. The full structure is suitable for use in ISO/IEC 15962 where the set of Object Identifiers to be encoded on the RFID tag have different higher level arcs, or where a generic system is in place.

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

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

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

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

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

1 0 15961 12 1

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

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

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

urn:oid:1.0.15961.12.1

6.3 Byte Notation

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

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

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

6.3.2 Bit ordering

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

Table 2 — Bit sequence in the 8-bit byte

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

6.3.3 Byte conversion

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

7 Processing application commands and responses

7.1 General

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

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

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

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

OPTION A: Straight through process

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

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

OPTION B: Using transfer encoding

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

7.2 Encoding system related information in commands

7.2.1 Singulation-Id

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

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

a) A completely Unique Item Identifier programmed in the RFID tag, as specified in the ISO/IEC 18000 series. In that series of standards it is defined as the Tag ID.

b) A data related identifier (e.g. like a Unique Item Identifier (UII)), that provides for uniqueness within the particular application domain of item management. This requires the UII to be read to establish the **Singulation-Id**.

c) A virtual or session identifier based on a time slot or other feature managed by the air interface protocol.

d) Combinations of (b) and (c), e.g. a virtual **Singulation-Id** across the air interface, but requiring the data related identifier to be returned as a response.

7.2.2 AFI

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

— If in the range 00_{16} to $0F_{16}$ the code applies to a closed system application, as defined in Part 3 of ISO/IEC 15961. The list of code values is updated in the associated RFID data constructs register.

— If in the range 90_{16} to CE_{16} the code applies to an open system application as defined in Part 3 of ISO/IEC 15961. The list of code values is updated in the associated RFID data constructs register.

— The value CF_{16} is reserved as an extension code for multiple byte **AFI** code values.

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

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

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

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

7.2.3 DSFID

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

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

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

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

7.2.4 Access-Method

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

0 **No-Directory** - This **Access-Method** provides the simplest set of encoding rules. The basic rule is to link a data **Object** to an **Object-Identifier**, and support this with appropriate syntactical components to create a Data-Set. The **Data-Sets** are concatenated to produce a contiguous sequence of encoded bytes in the Logical Memory Map.

1 **Directory** - The **Directory** structure supports all the encoding rules of the **No-Directory** structure, but additionally supports the encoding of a directory at the higher address values within the Logical Memory Map on the RFID tag. When the **Directory** structure is used, any command seeking to selectively read data can first access the directory to find the memory address of the beginning of the **Data-Set**, and then go to that location to read the bytes that represent the encoding of the **Data-Set**.

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

2 **Packed-Objects** - The **Packed-Object** structure has been introduced at the first revision of ISO/IEC 15962 and this edition of this part of ISO/IEC 15961. The encoding scheme is fundamentally different, because it takes a set of **Object-Identifiers** and their **Objects** and encodes them in an indexed structure that integrates compaction and encoding. The **Packed-Object** encoding scheme requires a rule-based table for each **Data-Format**, which calls up specific compaction schemes for the individual elements. The compaction schemes are standard, irrespective of the **Data-Format**. The **Packed-Object** scheme requires more complex encoding and decoding rules than the **No-Directory** and **Directory Access-Methods**, but offers significant encoding efficiency even over the basic **No-Directory** structure when a number of **Object-Identifiers** need to be encoded. This can reduce the amount of memory required on the RFID tag, and can reduce the size of message transferred across the air interface in response to a read command. Application domains need to define a table of indexed **Relative-OIDs** for this **Access-Method** to be implemented. It can only be applied as an alternative to the **No-Directory** or **Directory Access-Methods** on a particular RFID tag; but tags with any of the **Access-Methods** may be intermixed in the same application.

3 **Tag-Data-Profile** - The **Tag-Data-Profile** scheme is intended to support fixed message structures, typically where a number of **Object-Identifiers** and their **Objects** need to be encoded. The fixed message structure is best suited for applications that are reasonably homogeneous and additionally have a consistent requirement for exactly the same **Object-Identifiers** to be encoded on all RFID tags in the application. Each **Tag-Data-Profile** requires registration.

4 **Multiple-Records** - The **Multiple-Records** encoding process overlays a structure onto the **No-Directory**, **Packed-Objects**, and **Tag-Data-Profile Access-Methods** for multiple instances of these, and even a mixture of these **Access-Methods**, to be encoded in the same Logical Memory. This is achieved through the introduction of an encoded MR-header and a preamble for each record. This allows the encoding of individual records to fully comply with the inherent **Access-Method**. This also enables the same **Object-Identifier** to be encoded in separate records, with one rule that supports a list of the same data element in one record. There are three main classes of application supported by the **Access-Method** and these can also be intermixed. One class enables different data formats and owners of data to share the same RFID tag, but under the control of the original owner of the tag. Another supports a time sequence of history records of the same type to be repeated. The third major class id for the support of hierarchically related records e.g. for supply chain delivery or bill of material types of structure.

5 to 15 Reserved for future encoding schemes to be defined in ISO/IEC 15962 and invoked under rules for a multiple byte **DSFID**. Some of the new **Access-Methods** might be possible to be inherent encoding schemes in **Multiple-Records**. This can only be determined as the encoding schemes are developed.

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

— The **No-Directory** structure is better suited to RFID tags with small memory capacity, because the directory itself is an overhead that needs to be encoded. It also is better suited where there are few **Object-Identifiers** to be encoded, so that a continuous read function will transfer all the encoding (or at least sufficient of the encoded bytes) to enable the bytes to be parsed to find the required **Object-Identifier** and associated **Object.**

— The **Directory** structure is obviously better suited where the conditions differ from those suited to the **No-Directory** structure. In addition, it is better suited to applications that call for selective reading, writing, or modifying one or few **Object-Identifiers** from among many. In this type of situation, the extra read processes to transfer the directory to the Data Processor are likely to be balanced be the shorter read time for the selected **Object-Identifier**.

— The **Packed-Objects** structure requires more complex encoding and decoding, but offers a significant improvement in encoding efficiency over the basic **No-Directory** structure. As such, it is better suited to applications where the encoding requirements are relatively high compared to the memory capacity of the RFID tags used in the application.

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

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

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

7.2.5 Data-Format

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

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

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

0	**Not-Formatted** - This is used for RFID tags not formatted, or not yet formatted, to this part of ISO/IEC 15961.
1	**Full-Featured** - This **Data-Format** supports any type of data format where the full **Object-Identifier** is used. Its prime purpose is to enable heterogeneous data (i.e. from different data dictionaries) to be encoded on the one RFID tag. For example it could be used to encode data from different closed system applications unambiguously, using the ISO/IEC 9834-1 registered Object Identifiers for each application.
2	**Root-Oid-Encoded** - This **Data-Format** is used when all the data on the RFID tag has a common **Root-OID**, which does not comply with one of the specific **Root-OIDs** associated with a **Data-Format** registered under the rules of Part 2 of ISO/IEC 15961. The **Root-OID** for **Data-Format** 2 shall be directly encoded on the RFID tag using the appropriate root arcs. Each Object is encoded on the RFID tag using a **Relative-OID**, representing the remaining lower order arcs.
3 to 28	These code values applies to an open system application as registered under Part 2 of ISO/IEC 15961. The list of code values is updated in the associated RFID data constructs register.
29	This code applies to closed system data where the encoded data is compliant to the rules of this part of ISO/IEC 15961 and ISO/IEC 15962.
30	This code applies to closed system data, where the encoding rules are not compliant with ISO/IEC 15962. It is of use for applications that are migrating from other encoding rules, enabling data to be distinguished and properly encoded. It is also of use prior to the transition of RFID tags from a closed system application to an open system application.

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

7.3 Preparing the basic Objects and other application-based arguments

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

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

7.3.1 General model

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

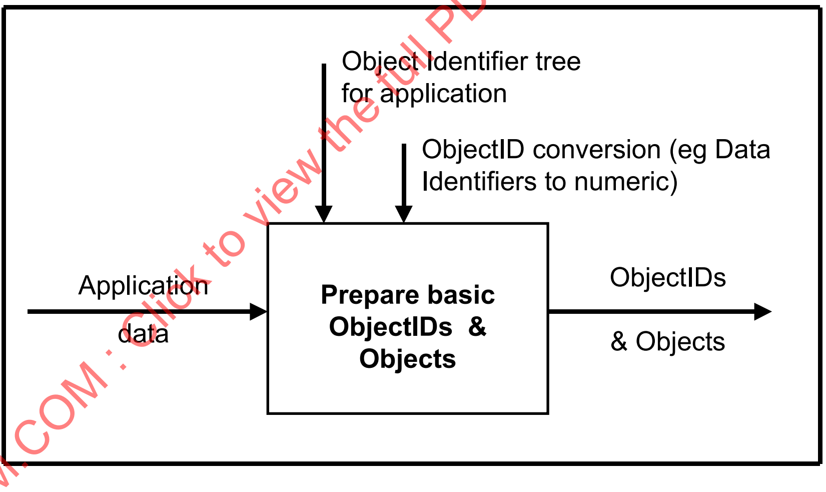


Figure 5 — Data flow model: Prepare basic Objects

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

7.3.2 Object-Identifier

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

7.3.3 Relating Object-Identifiers

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

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

7.3.4 Object

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

7.3.4.1 Basic 8-bit character set information

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

7.3.4.2 Support for ISO/IEC 10646

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

7.3.4.3 Support for secure or encrypted data

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

— It requires security of data to reside with the application and to be changed independently of the Data Processor.

— It allows the Data Processor to handle all data, whether encrypted or not, in a similar manner.

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

Further details are provided in 7.6.

7.3.5 Compact-Parameter

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

0 **Application-Defined** - The **Object** shall not be processed through the data compaction rules of ISO/IEC 15962 and remains unaltered when stored in the Logical Memory Map of the RFID tag.

1 **Compact** - This requires using the basic ISO/IEC 15962 compaction rules to compact the **Object** as efficiently as possible to reduce the number of bytes required on the Logical Memory Map.

2 **UTF8-Data** - This identifies that the **Object** has been externally transformed from the ISO/IEC 10646 coded character set to UTF-8 encoding. The **Object** shall not be processed through the data compaction rules of ISO/IEC 15962 and remains unaltered for transfer to the Logical Memory Map.

3 **Pack-Objects** - This identifies that a set of Objects is to be encoded using the Packed Object encoding scheme and identified with the associated **Access-Method**. The set of Objects shall not be compacted using the basic ISO/IEC 15962 compaction rules, but use the Packed Object encoding rules.

4 **Tag-Data-Profile** - This identifies that a set of **Objects** is to be encoded using the Tag Data Profile encoding scheme and identified with the associated **Access-Method**. Although the set of compaction schemes is identical to the basic ISO/IEC 15962 compaction rules, the Profile IDTable specifies a particular compaction scheme. The Profile IDTable specification shall be followed.

5 **Monomorphic-UII** - This identifies that a single **Object** defining a UII is to be encoded using the compaction rules defined by the **AFI**. As this is applied to a UII, this **Compact-Parameter** shall only be applied to a single **Object** per memory bank.

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

6 to 13 reserved for future definition

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

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

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

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

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

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

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

— If the command argument was **Application-Defined** (0), a compliant Data Processor undertook no compaction and applied the relevant compact code to the encoding on the tag. As the decoder cannot interpret the encoded bytes, the response is **Application-Defined** (0).

— If the command argument was **Compact** (1) a compliant Data Processor applied the correct compact code to the encoding on the tag. As the decoder can interpret the encoded bytes, the response is **De-Compacted-Data** (15).

— If the command argument was **UTF8-Data** (2) a compliant Data Processor undertook no compaction and applied the relevant compact code to the encoding on the tag. As the decoder cannot interpret the encoded bytes, the response is **UTF8-Data** (2).

— If the command argument was **Pack-Objects** (3) a compliant Data Processor encoded to the rules as defined for the particular **Object** as defined by the ID Table specified. The decoder uses the same ID table to interpret the encoded byte and returns each data **Object** with the response **De-Compacted-Data** (15).

— If the command argument was **Tag-data-Profile** (4) a compliant Data Processor encoded to the rules as defined for the particular **Object** as defined by the ID Table specified. The decoder uses the same ID table to interpret the encoded byte and returns each data **Object** with the response **De-Compacted-Data** (15).

— If the command argument was **Monomorphic-UII** (5) a compliant Data Processor encoded to the rules as defined for the particular **Object** as defined by the **AFI** and the Data constructs register. The decoder, on recognition of the AFI uses the compaction rule declared on the Data Constructs register and returns the data **Object** with the response **De-Compacted-Monomorphic-UII** (14). It also returns the **Object-Identifier** as defined by the Data Constructs register.

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

7.3.6 Object-Lock

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

7.4 Other command arguments

7.4.1 Access-Password

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

7.4.2 Additional-App-Bits

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

7.4.3 AFI-Lock

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

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

7.4.4 Append-To-Existing-Multiple-Record

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

7.4.5 Application-Defined-Record-Capacity

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

7.4.6 Avoid-Duplicate

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

7.4.7 Battery-Assist-Indicator

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

7.4.8 Block-Align

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

7.4.9 Block-Align-Packed-Object

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

7.4.10 Check-Duplicate

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

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

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

7.4.11 Data-CRC-Indicator

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

7.4.12 Data-Length-Of-Record

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

7.4.13 Delete-MR-Method

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

0 Mark record as deleted
1 Physically delete the record

The process details are defined in 10.27.1.

7.4.14 Directory-Length-EBV8-Indicator

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

1 Single byte EBV-8, which can support a directory of up to 127 blocks.

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

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

7.4.15 DSFID-Lock

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

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

7.4.16 DSFID-Pad-Bytes

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

7.4.17 Editable-Pointer-Size

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

7.4.18 Encoded-Memory-Capacity

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

7.4.19 EPC-Code

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

7.4.20 Full-Function-Sensor-Indicator

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

7.4.21 Hierarchical-Identifier-Arc

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

7.4.22 Identifier-Of-My-Parent

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

7.4.23 Identify-Method

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

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

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

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

7.4.24 ID-Type

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

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

7.4.25 Instance-Of-Arc

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

7.4.26 Kill-Password

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

7.4.27 Length-Of-Mask

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

7.4.28 Lock-Directory-Entry

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

7.4.29 Lock-Multiple-Records-Header

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

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

7.4.30 Lock-Record-Preamble

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

7.4.31 Lock-UII-Segment-Arguments

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

7.4.32 Max-App-Length

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

7.4.33 Memory-Bank

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

7.4.34 Memory-Bank-Lock

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

7.4.35 Memory-Length-Encoding

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

7.4.36 Memory-Segment

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

7.4.37 Memory-Type

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

7.4.38 Multiple-Records-Directory-Length

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

7.4.39 Multiple-Records-Features-Indicator

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

— Bit 8 with the value 1_2 indicates that all the records use the same **Access-Method**. If this bit = 0 then bits 7 to 4 = 0000_2

— Bits 7 to 4 identify the particular **Access-Method** if this is applied consistently. If bit 8 = 0, the currently permissible string for bits 7 to 4 are:

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

— Bit 3 with the value 1 indicates that some of the records are in a hierarchical relationship with other records.

— Bit 2 identifies whether the number of records field is fully maintained (e.g. updated each time a new record is added) or whether the number of records might not be the correct current value. The value 1 indicates that the number of records is correct, the value 0 indicates that the number of records field might be incorrect.

— Bit 1 with the value 1 indicates that there is an additional byte for indicating additional features. Currently this is RFU and therefore set to 0.

7.4.40 NSI-Bits

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

7.4.41 Number-In-Data-Element-List

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

7.4.42 Number-Of-Records

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

0 then this field is not maintained and the value of **Number-Of-Records** should be zero, but any other value should be ignored, for example if some record count was initially maintained but later this function was decided to be stopped.

1 then this field is fully maintained as new records are added. A zero value for the **Number-Of-Records** is encoded when the MR-header is created.

7.4.43 Number-Of-Tags

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

7.4.44 Objects-Offsets-Multiplier

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

7.4.45 Packed-Object-Directory-Type

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

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

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

7.4.46 Password

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

7.4.47 Password-Type

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

7.4.48 PO-Directory-Size

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

7.4.49 PO-Index-Length

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

7.4.50 Pointer

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

7.4.51 Pointer-To-Multiple-Records-Directory

The command and response argument **Pointer-To-Multiple-Records-Directory** is used to define the highest block number at the start address of the directory. This is defined as an EBV-8 value. If a directory is not initially encoded, then a fixed length EBV-8 string of the same length as required for the start point of the directory shall be encoded with a value zero. The directory is encoded in reverse block sequence so that the next subsequent block of the directory is at the next lower address. The start of the directory is not necessarily the highest address in Logical Memory, because other hardware features of the tag might be specified to be located there. It will be necessary to invoke the air interface commands that define the memory mapping to determine the start address of the directory.

7.4.52 Read-Record-Type

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

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

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

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

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

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

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

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

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

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

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

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

7.4.54 Record-Memory-Capacity

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

7.4.55 Record-Type-Arc

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

7.4.56 Record-Type-Classification

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

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

7.4.57 Sector-Identifier

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

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

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

7.4.58 Simple-Sensor-Indicator

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

7.4.59 Start-Address-Of-Record

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

7.4.60 Tag-Data-Profile-ID-Table

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

7.4.61 Tag-Mask

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

7.4.62 Update-Multiple-Records-Directory

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

— If a directory pre-exists and the command argument is set to TRUE, the directory is fully updated, including any previously missed directory entries.

— If a directory pre-exists and the command argument is set to FALSE, the argument is ignored and the directory is fully updated, including any previously missed directory entries.

— If no directory exists and the command argument is set to TRUE, the directory is created and fully updated, including any previously missed directory entries.

— If no directory exists and the command argument is set to FALSE, no directory is created.

7.4.63 Word-Count

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

7.4.64 Word-Pointer

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

7.5 Command-related field names

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

7.5.1 Data-Set

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

7.5.2 Identities

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

7.5.3 Length-Lock Byte

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

7.5.4 Length-Of-Encoded-Data

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

7.5.5 Lock-Status

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

7.5.6 Logical-Memory-Map

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

7.5.7 Memory-Capacity

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

7.5.8 Module-OID

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

7.5.9 Number-Of-Tags-Found

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

7.5.10 PO-ID-Table

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

7.5.11 Protocol-Control-Word

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

7.5.12 Read-Data

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

7.6 Data security

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

The following features can be provided:

— Data security - This allows only authorised users the ability to "see" the true data. The sender will need to provide authorised users with the decryption algorithm and key. Other users can see the raw octet string, but this will be meaningless.

— Data integrity - This uses the decrypted data to identify whether the data has been modified by others prior to being read by an authorised user.

— Data validity - This uses encryption processes to reduce the risk of data from a legitimate source being copied by an unauthorised agent into another RFID tag, and being passed off as the original source RFID tag.

Additional advice is provided in Annex D.

8 Data flows and processes to the air interface

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

8.1 Establishing communications between the application and the RFID tag

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

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

8.1.1 Air interface services

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

— Application memory is an integer number of bytes.

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

— Application memory shall be organised in blocks. These shall be fixed size and be of one or more bytes.

— The individual blocks shall each be accessible by read and/or write.

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

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

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

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

8.1.2 System information

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

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

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

physical block size
number of blocks

8.2 Application system services

The application system shall provide the following:

— **Object Identifier** - (see 7.3.2)

— **Object** - (see 7.3.4)

— **Compact-Parameter** - (see 7.3.5)

— **Object-Lock** - (see 7.3.6)

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

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

Application commands are used to instruct the Data Processor and the interrogator to execute specific functions. They are also applied to the processing of the application data to achieve efficient encoding. The responses from the Data Processor include requested data and also information about actions undertaken and errors found. Each command/ response pair has its abstract syntax presented as modules. Each module is defined in a manner that enables the command to be invoked independently of any other command.

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

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

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

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

9.1 Final arc values of the command and response modules

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

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

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

9.2 Completion-Code

The **Completion-Code** is part of the response to each command. The **Completion-Code** is an INTEGER value that reports specifically on how the particular command was processed and executed, successfully or not. It is returned in each response. If its value is 0 (00_{16}), the command has been successfully executed. If its value is 255 (FF_{16}), the command could not be executed by the system for the reason specified in the **Execution-Code** (see 9.3). If its value is different from 0 and 255, it indicates that the command was not executed as instructed by the application for the reason mentioned.

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

The following **Completion-Codes** apply:

0 **No-Error:** The command was successfully executed.

1 **AFI-Not-Configured:** The command could not be completed, possibly because of a prior configure action.

2 **AFI-Not-Configured-Locked:** The **AFI** was found to be locked (by a previous command execution), so this **AFI** could not be configured, as requested by the command, on this occasion.

3 **AFI-Configured-Lock-Failed:** The **AFI** was correctly configured and encoded, but the lock function could not be completed.

4 **DSFID-Not-Configured:** The command could not be completed, possibly because of a prior configure action.

5 **DSFID-Not-Configured-Locked:** The **DSFID** was found to be locked (by a previous command execution), so this **DSFID** could not be configured, as requested by the command, on this occasion.

6 **DSFID-Configured-Lock-Failed:** The **DSFID** was correctly configured and encoded, but the lock function could not be completed.

7 **Object-Locked-Could-Not-Modify:** The existing encodation on the RFID tag is locked, and as a result this attempt to update the **Object** value could not be completed.

8 **Singulation-Id-Not-Found:** The particular RFID tag, specified by the **Singulation-Id**, could not be found in the operating area.

9 **Object-Not-Added:** The data set of Precursor, **Object** and **Object-Identifier** could not be added to the Logical Memory Map (e.g. because there was insufficient memory space). This response also applies if the RFID tag supports a lock function that failed.

10 **Duplicate-Object:** An **Object-Identifier** with the same value as the one to be processed (added, modified, or deleted) was found. The process was aborted.

11 **Object-Added-But-Not-Locked:** The data set was added to a RFID tag that did not support a lock feature in the memory.

12 **Object-Not-Deleted:** The data set of Precursor, **Object-Identifier** and **Object** could not be deleted from the Logical Memory Map (e.g. because the delete function is not supported by the particular RFID tag).

13 **Object-Identifier-Not-Found:** The intended process could not be executed because the defined **Object-Identifier** is not actually encoded on the RFID tag.

14 **Object-Locked-Could-Not-Delete:** The **Object** is already locked and so cannot be deleted.

15 **Object-Not-Read:** The intention to read the specified **Object** failed.

16 **Objects-Not-Read:** The intention to read one or more of the specified **Objects** failed.

17 **Blocks-Locked:** The intended action to erase encoded bytes from one or more blocks could not be achieved because blocks were previously locked.

18 **Erase-Incomplete:** The intended action to erase encoded bytes from one or more blocks was interrupted and that not all blocks have been processed. The command can be re-invoked to complete the process.

19 **Read-Incomplete:** The intention to read the complete contents of the Logical Memory Map failed, because not all blocks from the RFID tag were transferred to the Logical Memory.

20 **System-Info-Not-Read:** The intention to read the system information failed.

21 **Object-Not-Modified:** The data set of Precursor, **Object-Identifier** and **Object** could not be modified on the Logical Memory Map (e.g. because the modify function is not supported by the particular RFID tag).

22 **Object-Modified-But-Not-Locked:** The **Object** was modified in a RFID tag that did not support a lock feature in the memory.

23 **Failed-To-Read-Minimum-Number-Of-Tags:** The minimum number of RFID tags were not identified with the specified selection criterion, possibly because of a time-out.

24 **Failed-To-Read-Exact-Number-Of-Tags:** The exact number of RFID tags were not identified with the specified selection criterion, possibly because of a time-out.

25 **Password-Mismatch:** The **Password** presented in the command did not match the equivalent password on the RFID tag, so the associated action was aborted.

26 **Password-Not-Written:** The **Password** was not written, either because the memory location was already encoded, or the feature was not supported.

27 **Zero-Kill-Password-Error:** The Kill operation was not executed because the **Kill-Password** has a zero value.

28 **Kill-Failed:** The intention to render the tag failed (e.g. because of insufficient power).

29 **Object-Not-Editable**: The **Object**, encoded as part of a **Packed-Object**, was not modified because it is not editable.

30 **Directory-Already-Defined**: A **Packed-Object** already has a directory type defined.

31 **Packed-Object-ID-Table-Not-Recognised**: The ID-Table called for in the command was not supported by the encoder or interrogator, and encoding is impossible to achieve.

32 **Tag-Data-Profile-ID-Table-Not-Recognised**: The ID-Table called for in the command was not supported by the encoder or interrogator, and encoding is impossible to achieve.

33 **Insufficient-Tag-Memory**: The operation failed because there was insufficient memory on the tag to satisfy the requested operation.

34 **AFI-Not-For-Monomorphic-UII**: The operation failed because the **AFI** called for in the command is not registered for **Monomorphic-UIIs**.

35 **Monomorphic-UII-OID-Mismatch**: The **Object-Identifier** defined in the command does not match that on the ISO?IEC 15961-2 Data Constructs register for this AFI.

36 **Command-Cannot-Process-Monomorphic-UII**: A **Monomorphic-UII** was presented as an **Object** to command that does support this type of UII.

37 **Data-CRC-Not-Applied**: At least one of the **Data-CRCs** requested in the **Extended-DSFID** was not applied to the data.

38 **Length-Not-Encoded-In-DSFID**: At least one of the requested lengths was not encoded in the **Extended-DSFID**.

39 **Multiple-Records-Header-Not-Configured**: Some of the processes required to configure the **Multiple-Records-Header** could not be processed by the Data Processor, resulting in an incomplete record.

40 **Multiple-Records-Header-Not-Locked**: The **Multiple-Records-Header** was added to an RFID tag that did not support a selective lock feature in the memory.

41 **File-Support-Indicators-Not-Configured**: Some of the processes required to configure the **File-Support-Indicators** could not be processed by the Data Processor, resulting in an incomplete encoding.

42 **File-Support-Indicators-Not-Locked**: The **File-Support-Indicators** field was added to an RFID tag that did not support a selective lock feature in the memory.

43 **Data-Format-Not-Compatible-Multiple-Records-Header**: The **Data-Format** in the command is at variance with the controlling **Data-Format** rules defined in the MR-header. The record was not encoded.

44 **Access-Method-Not-Compatible-Multiple-Records-Header**: The **Access-Method** in the command is at variance with the controlling **Access-Method** rules defined in the MR-header. The record was not encoded.

45 **Sector-Identifier-Not-Compatible-Multiple-Records-Header**: The sector identifier in the **Object-Identifier** in the command is at variance with the controlling sector identifier rules defined in the MR-header. The record was not encoded.

46 **Record-Preamble-Not-Configured**: Some of the processes required to configure the record preamble could not be processed by the Data Processor, resulting in an incomplete encoding.

47 **Record-Preamble-Not-Locked**: The record preamble was added to an RFID tag that did not support a selective lock feature in the memory.

48 **Multiple-Records-Directory-Not-Present**: The command to read this directory could not be invoked because of the absence of the directory.

49 **Record-Not-Deleted-Preamble-Locked**: The command could not be invoked because the record's preamble is locked.

50 **Record-Not-Deleted-Directory-Locked**: The command could not be invoked because the record's entry in the directory is locked.

51 **Record-Not-Deleted-Lower-Level-Preamble-Locked**: The command could not be invoked because the preamble of a lower-level record is locked.

52 **Record-Not-Deleted-Encoding-Locked**: the command could not be invoked because part of the record is locked.

253 **ISO/IEC 24791-5 Result-Code:** This **Completion-Code** is used to pass through the Result code from an ISO/IEC 24791-5 response in the form of {message code} {result value}.

254 **Undefined-Command-Error:** An error occurred in a command, not defined by another **Completion-Code**.

255 **Execution-Error:** A system error occurred which made it impossible to action the command. The appropriate **Execution-Code** is returned in the response.

9.3 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 following **Execution-Codes** apply:

0 **No-Error**: The command was executed without error

1 **No-Response-From-Tag**: No response was received from the RFID tag

2 **Tag-Communication-Error**: The response(s) from the RFID tag(s) was corrupted (e.g. There was an aborted frame)

3 **Tag-CRC-Error**: A CRC error was detected in the RFID tag's response

4 **Command-Not-Supported**: The command code is not supported by the interrogator, or RFID tag

5 **Invalid-Parameter**: The command parameter(s) are invalid

6 **Interrogator-Communication-Error**: An error occurred in the communication between the application and in the interrogator

7 **Internal-Error**: An error occurred in the application software

255 **Undefined-Error**: An error occurred, not defined by another code

10 Commands and responses

Each pair of command and response is specified in the sub-clauses 10.1 to 10.21. Those arguments that require a separate specification are defined in 10.27.2.

10.1 Configure-AFI

10.1.1 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 particular applications are available on the Register of Data Constructs provided by the Registration Authority of ISO/IEC 15961-2. 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, but 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:

Value	Definition
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
CF_{16}	Reserved as an extension code

AFI-Lock: BOOLEAN
If set to TRUE, the interrogator shall lock the AFI

10.1.2 Configure-AFI response

The **Configure-AFI** response has the following field names:

Completion-Code (see 9.2)
Execution-Code (see 9.3)

Configure-AFI response

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

Completion-Code: INTEGER

Possible Values:

Value	Definition
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 9.3.

10.2 Configure-DSFID

Configure-Extended-DSFID (see 10.24) is required when encoding indicators of other features in the RFID tag, such as the use of a **Data-CRC**. That command might also be better to encode an **Access-Method** with a value greater than 3, or a **Data-Format** with a value greater than 31.

10.2.1 Configure-DSFID command

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

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 0)
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

10.2.2 Configure-DSFID response

The **Configure-DSFID** response has the following field names:

Completion-Code (see 9.2)
Execution-Code (see 9.3)

Configure-DSFID response

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

Completion-Code: INTEGER
Possible Values:

Value	Definition
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 9.3.

10.3 Inventory-Tags

10.3.1 Inventory-Tags command

The **Inventory-Tags** command requires that the value of the **AFI** is specified to select RFID 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 RFID 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 RFID tags, complying 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 RFID tag entering the operating area, is to set the **Inventory-At-Least** field to 1.

Particular 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)

Possible Values:

Value	Definition
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 RFID 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 RFID 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 will wait until the first RFID 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 will wait until the specified number of RFID tags has been detected.

If the **Identify-Method** is set to **Inventory-No-More-Than**, the interrogator shall initiate an inventory of the RFID 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 till all tags have been read.

NOTE This may be constrained by the air interface and anticollision mechanism.

If the **Identify-Method** is set to **Inventory-Exactly**, the interrogator shall initiate an inventory of the RFID 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 could 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 This may be constrained by the air interface and anticollision mechanism.

Execution of this command with the arguments **Inventory-At-Least** and **Inventory-Exactly** can cause the interrogator to wait until sufficient RFID 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.

10.3.2 Inventory-Tags response

The **Inventory-Tags** response has the following field names:

Completion-Code (see 9.2)
Execution-Code (see 9.3)
Identities (see 7.5.2)
Number-Of-Tags-Found (see 7.5.9)

Inventory-Tags response

Module-OID: OBJECT IDENTIFIER = 1 0 15961 127 3

Completion-Code: INTEGER
Possible Values:

Value	Definition
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
Possible Values: As defined in 9.3.

Number-Of-Tags-Found: INTEGER (1..65535)

Identities: List of <Singulation-Id>

10.4 Delete-Object

10.4.1 Delete-Object command

The **Delete-Object** command instructs the interrogator to delete a defined **Object-Identifier** and its **Object** and associated parameters. Only one RFID 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 RFID 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 10.27.

The **Delete-Object** command instructs the interrogator to delete the data set specified by its **Object-Identifier** from the RFID tag Logical Memory Map. This procedure might not succeed if the data set is locked, if this is found to be the case, the response will return 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 RFID 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**.

NOTE 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 (see7.3.2)
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 Object-Identifier

10.4.2 Delete-Object response

The **Delete-Object** response has the following field names:

Completion-Code (see 9.2)
Execution-Code (see 9.3)

Delete-Object response

Module-OID: OBJECT IDENTIFIER = 1 0 15961 127 5

Completion-Code: INTEGER

Possible Values:

Value	Definition
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

Possible Values: As defined in 9.3.

10.5 Modify-Object

10.5.1 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 RFID tag. The complete memory needs to 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 RFID 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 RFID 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 10.23.

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

1. Read the complete Logical Memory Map from the RFID tag.
2. 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.
3. 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**

If the **Object** is already locked, it cannot be modified and the appropriate completion code shall be returned.

If the **Access-Method** is **Packed-Objects,** and the **Packed-Object** is not editable, the **Object** cannot be modified and the appropriate completion code shall be returned.

If the **Access-Method** is **Tag-Data-Profile**, three conditions can be presented:

- if the new compacted data is the same length, then this data is simply overwritten
- if it is shorter, it is written to the tag with the necessary pad bytes
- if it is longer, there is an error and the data cannot be modified and the appropriate completion code shall be returned.

If the data object is part of a multiple record (identified with the basic root-OID of **1.0.15961.401**) or a hierarchical multiple record (identified with the basic root-OID of **1.0.15961.402**), then the process to modify a data object shall be that of the declared **Access-Method**. A data object cannot be modified on a data element list (identified with the basic root-OID of **1.0.15961.403**).

The **Modify-Object** command has the following arguments:

Compact-Parameter (see 7.3.5)
Object (see 7.3.4)
Object-Identifier (see7.3.2)
Object-Lock (see 7.3.6)
Singulation-Id (see 7.2.1)

Modify-Object command

Module-OID: OBJECT IDENTIFIER = 1 0 15961 126 6

Singulation-Id: INTEGER (0..255)

Object-Identifier: OBJECT IDENTIFIER

Object: BYTE STRING

Compact-Parameter: INTEGER (0..15)

Possible Values:

Value	Definition (see 7.3.5 for further details)
0	Application-Defined
1	Compact
2	UTF8-Data
3	Packed-Objects
4	Tag-Data-Profile

Object-Lock: BOOLEAN
If TRUE the interrogator shall lock the relevant Data-Set

10.5.2 Modify-Object response

The **Modify-Object** response has the following field names:

Completion-Code (see 9.2)
Execution-Code (see 9.3)

Modify-Object response

Module-OID: OBJECT IDENTIFIER = 1 0 15961 127 6

Completion-Code: INTEGER

Possible Values:

Value	Definition
0	No-Error
7	Object-Locked-Could-Not-Modify
8	Singulation-Id-Not-Found
10	Duplicate-Object
13	Object-Identifier-Not-Found
21	Object-Not-Modified
22	Object-Modified-But-Not-Locked
33	Insufficient-Tag-Memory
36	Command-Cannot-Process-Monomorphic-UII
37	Data-CRC-Not-Applied
38	Length-Not-Encoded-In-DSFID
255	Execution-Error

Execution-Code: INTEGER

Possible Values: As defined in 9.3.

10.6 Read-Object-Identifiers

10.6.1 Read-Object-Identifiers command

This command shall not be used with Multiple Records. Instead the **Read-Multiple-Records** command shall be used (see 10.26).

The **Read-Object-Identifiers** command instructs the interrogator to read all the **Object-Identifiers** from the RFID tag. This module can be used in advance of a more selective command to read a specific **Object**, or to identify duplicate **Object-Identifiers** so that a housekeeping procedure can be invoked. A valid response, if the RFID tag Logical Memory Map has no **Object-Identifiers** stored, is to return an empty **Object-Identifiers** list. Only one RFID tag shall be programmed per command to ensure that the read process is robust.

The **Read-Object-Identifiers** command has the following argument:

Singulation-Id (see 7.2.1)

Read-Object-Identifiers command

Module-OID: OBJECT IDENTIFIER = 1 0 15961 126 8

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

10.6.2 Read-Object-Identifiers response

The **Read-Object-Identifiers** response has the following arguments:

Completion-Code (see 9.2)
Execution-Code (see 9.3)
Read-OIDs-Response-List (see 0)

Read-Object-Identifiers response

Module-OID: OBJECT IDENTIFIER = 1 0 15961 127 8

Read-OIDs-Response-List: List of <Read-OIDs-Response>

Completion-Code: INTEGER

Possible Values:

Value	Definition
0	No-Error
8	Singulation-Id-Not-Found
255	Execution-Error

Execution-Code: INTEGER

Possible Values: As defined in 9.3.

10.7 Read-Logical-Memory-Map

10.7.1 Read-Logical-Memory-Map command

The main function of this command is for diagnostic purposes, but it can also be used for other functions where reading the complete content of the Logical Memory Map is required. Only one RFID tag should be programmed per command to ensure that the reading process is robust.

The **Read-Logical-Memory-Map** command instructs the interrogator to read the entire Logical Memory Map of the RFID tag and respond with this without any decoding and interpretation (i.e. by returning the encoded byte values). No processing takes place through the Data Processor as part of this read command, so that individual **Object-Identifiers, Objects, Compact-Parameter** and **Lock-Status** cannot be directly identified.

The command applies equally to all **Access-Methods**, but if a **Directory** structure has been defined by the **Access-Method,** this shall be included in the response, but shall not be distinguished from other bytes in the Logical Memory Map.

The **Read-Logical-Memory-Map** command has the following argument:

Singulation-Id (see 7.2.1)

Read-Logical-Memory-Map command

Module-OID: OBJECT IDENTIFIER = 1 0 15961 126 10

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

10.7.2 Read-Logical-Memory-Map response

The **Read-Logical-Memory-Map** response has the following field names:

Completion-Code (see 9.2)
Execution-Code (see 9.3)
Logical-Memory-Map (see 7.5.6)

Read-Logical-Memory-Map response

Module-OID: OBJECT IDENTIFIER = 1 0 15961 127 10

Completion-Code: INTEGER
Possible Values:

Value	Definition
0	No-Error
8	Singulation-Id-Not-Found
19	Read-Incomplete
255	Execution-Error

Execution-Code: INTEGER
Possible Values: As defined in 9.3.

Logical-Memory-Map: BYTE STRING

10.8 Erase-Memory

10.8.1 Erase-Memory command

The **Erase-Memory** command instructs the interrogator to re-set to zero the entire Logical Memory Map of the specified RFID tag. This includes the **Directory**, if this is defined as the **Access-Method**. If none of the blocks is locked, this should result in a deletion of all **Data-Sets** or **Packed-Objects**. If any block is locked, then the **Completion-Code**: **Blocks-Locked** will be returned. Only one RFID tag shall be programmed per command to ensure that the erasure process is robust.

Subsequent processing by the application, possibly by reading all the locked data, can be invoked to determine whether the RFID tag is still useable. For example, the blocks that are locked could contain data permanently assigned to the item and the unlocked blocks containing transitory data.

The **Erase-Memory** command has the following argument:

Singulation-Id (see 7.2.1)

Erase-Memory command

Module-OID: OBJECT IDENTIFIER = 1 0 15961 126 12

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

10.8.2 Erase-Memory response

The **Erase-Memory** response has the following field names:

Completion-Code (see 9.2)
Execution-Code (see 9.3)

Erase-Memory response

Module-OID: OBJECT IDENTIFIER = 1 0 15961 127 12

Completion-Code: INTEGER

Possible Values:

Value	Definition
0	No-Error
8	Singulation-Id-Not-Found
17	Blocks-Locked
18	Erase-Incomplete
255	Execution-Error

Execution-Code: INTEGER

Possible Values: As defined in 9.3.

10.9 Get-App-Based-System-Info

10.9.1 Get-App-Based-System-Info command

The **Get-App-Based-System-Info** command instructs the interrogator to read the system information and return those arguments that are relevant to the application, namely the **AFI** and **DSFID**. This command is useful for RFID tag types that do not return these codes as part of a response to other commands.

The **Get-App-Based-System-Info** command has the following argument:

Singulation-Id (see 7.2.1)

Get-App-Based-System-Info command

Module-OID: OBJECT IDENTIFIER = 1 0 15961 126 13

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

10.9.2 Get-App-Based-System-Info response

The **Get-App-Based-System-Info** response has the following arguments:

AFI (see 7.2.2)
Completion-Code (see 9.2)
DSFID (see 7.2.3)
Execution-Code (see 9.3)

Get-App-Based-System-Info response

Module-OID: OBJECT IDENTIFIER = 1 0 15961 127 13

AFI: BYTE

DSFID: BYTE STRING

Completion-Code: INTEGER

Possible Values:

Value	Definition
0	No-Error
8	Singulation-Id-Not-Found
20	System-Info-Not-Read
255	Execution-Error

Execution-Code: INTEGER

Possible Values: As defined in 9.3.

10.10 Write-Objects

10.10.1 Write-Objects command

The **Write-Objects** command is used to write one or more **Object-Identifiers** and associated **Objects** to an RFID tag. This command may be implemented to write the initial data to the RFID tag, or to add data to the tag. The command is supported by a compound argument **Add-Objects-List**.

This command shall not be used to write a **Monomorphic-UII**. If the AFI on the RFID 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 10.23.

The **DSFID-Constructs-list** is used to specify the **Access-Method** and **Data-Format**. The **Ext-DSFID-Constructs-list** is used to set indicators on the **Extended-DSFID** and to instruct the Data Processor to carry out certain procedures, e.g. to apply a CRC to the data.

The **DSFID-Constructs-list** and the **Ext-DSFID-Constructs-list** are provided for use in one of the following ways:

— If data is being written to a blank RFID tag, then they are provided as part of this command to minimise communications.

— If data is being added to the RFID tag, then the **DSFID** in the command should match the **DSFID** already encoded on the RFID tag, else there is an error and the encoding process can cease before significant amounts of data have been processed.

— Additionally all the requirements declared by the arguments in the **Ext-DSFID-Constructs-list** shall be processed.

The **DSFID-Lock** argument, if set, is applied to the entire **DSFID** and **Extended-DSFID** byte string.

If the **Access-Method** is **Packed-Objects**, all **Objects** specified as arguments shall be included within the same new **Packed-Object** added after any existing **Packed-Objects** in memory. A number of arguments only apply to **Packed-Objects** and these are defined in the **Packed-Object-Constructs** argument.

If the **Access-Method** is **Tag-Data-Profile**, all **Objects** specified as arguments shall be included within the same **Tag-Data-Profile**. Adding an Object that is not specified by the **Tag-Data-Profile-ID-Table** argument shall be treated as an error, and no encoding take place.

A **Multiple-Record** is declared by the **Object-Identifiers** in the **Add-Objects-List** all being in one of these three mandatory forms:

— **1.0.15961.401. {data format = dictionary}.**{sector identifier}.{record type}.{instance-of}.{Relative-OID of data element}

— **1.0.15961.402.{data format = dictionary}.**{sector identifier}.{record type}.{hierarchical id}.{ Relative-OID of data element}

— **1.0.15961.403.{data format = dictionary}.**{sector identifier}.{record type}.{hierarchical id}.{ Relative-OID of data element}.{list element number}

The **Add-Objects-List** argument applies to a single record in a multiple records structure. Therefore in the OID structures listed above only the final arc values are permitted to differ in a single command.

Individual records shall be written only after the MR-header has been created.

The **DSFID-Constructs-List** argument shall be used to declare that the individual Multiple-Record complies with the encoding rules of one of the following **Access-Methods**:

0	No-Directory
2	Packed-Objects
3	Tag-Data-Profile

For multiple records, the command requires the **Multiple-Records-Constructs-List** arguments (see 11.8) to be defined. This includes instructions about reserving memory to enable the record to increase in size, to indicate whether a directory entry is required, and in some cases to identify whether there is a parent-child relationship between this record and others. There is also an argument to declare whether this record is defined as a data element list, in which case the **Add-Objects-List** contains multiple instances of the same **Relative-OID**. The command may also be used to add data elements to an existing multiple record by declaring this in the relevant argument in the **Multiple-Records-Constructs-List.**

The **Ext-DSFID-Constructs-List** is also required for defining all the encoded requirements for a **Multiple-Record**.

The MR-header defines rules for the **Data-Format**, **Access-Method** and sector identifier that govern the subsequent encoding of individual records. Depending on their individual settings, the rules either required the associated argument to be the same for all records or to be permitted to be different. Any variance with the MR-header shall case the command to be aborted and the record not encoded.

The **DSFID-Lock** argument is not relevant for multiple records because this is part of the record preamble then needs to be considered as a unit for locking or not locking.

The **Write-Objects** command has the following arguments:

Add-Objects-List (see 11.1)
DSFID-Constructs (see 11.2)
DSFID-Lock (see 0)
Ext-DSFID-Constructs (see 11.4)
Multiple-Records-Constructs (see 11.8)
Packed-Object-Constructs (see 11.12)
Tag-Data-Profile-ID-Table (see 7.4.60)
Singulation-Id (see 7.2.1)

Write-Objects command

Module-OID: OBJECT IDENTIFIER = 1 0 15961 126 17

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

DSFID-Constructs-list: [OPTIONAL] List of <DSFID-Constructs>

Ext-DSFID-Constructs-list: [OPTIONAL] List of <Ext-DSFID-Constructs>

DSFID-Lock: BOOLEAN
If set to TRUE, the interrogator shall lock the DSFID

Add-Objects-List: List of <Add-Objects>

Packed-Object-Constructs: [OPTIONAL] List of <Packed-Object-Constructs>

Tag-Data-Profile-ID-Table: INTEGER [OPTIONAL]

Multiple-Records-Constructs: [OPTIONAL] List of <Multiple-Records-Constructs>

10.10.2 Write-Objects response

The **Write-Objects** response has the following field names:

Completion-Code (see 9.2)
Execution-Code (see 9.3)
Write-Responses (see 11.18)

Additional **Completion-Codes** apply to each **Object-Identifier** and are incorporated in the **Write-Response** argument.

Write-Objects response

Module-OID: OBJECT IDENTIFIER = 1 0 15961 127 17

Write-Responses-List: List of <Write-Responses>

Completion-Code: INTEGER
Possible Values:

Value	Definition
0	No-Error
8	Singulation-Id-Not-Found
29	Object-Not-Editable
31	Packed-Object-ID-Table-Not-Recognised-No-Encoding
32	Tag-Data-Profile-ID-Table-Not-Recognised
33	Insufficient-Tag-Memory
36	Command-Cannot-Process-Monomorphic-UII
37	Data-CRC-Not-Applied
38	Length-Not-Encoded-In-DSFID
43	Data-Format-Not-Compatible-Multiple-Records-Header
44	Access-Method-Not-Compatible-Multiple-Records-Header
45	Sector-Identifier-Not-Compatible-Multiple-Records-Header
46	Record-Preamble-Not-Configured
47	Record-Preamble-Not-Locked
255	Execution-Error

Additional Completion-Codes apply to the Write-Response-List

Execution-Code: INTEGER
Possible Values: As defined in 9.3.

10.11 Read-Objects

10.11.1 Read-Objects command

This command shall not be used to read any data object or other part of a multiple record. The relevant procedures are defined in 10.26.

The **Read-Objects** command instructs the interrogator to read a set of one or more **Object-Identifiers** and associated **Objects** from the RFID tag. A command argument can be used to check that the **Object-Identifier** is not duplicated on the RFID tag. Only one RFID tag shall be programmed per command to ensure that the reading process is robust.

The command supports an argument that enables the application to prescribe an upper address point on the RFID tag beyond which reading is discontinued. This argument incorporates the features of the original **readFirstObject** command, but is no longer restricted to just one **Object-Identifier**. This command argument supports features across the air interface that could be faster than reading a named **Object-Identifier(s)**. The application may therefore select to have the most often accessed **Object(s)** stored first in the tag. A value needs to be determined (in bytes) for the **Max-App-Length** argument, which can be achieved be simulating the encoding or by a short trial period, changing the value of this argument until the command achieves a high probability of reading the requested **Object-Identifier(s)** and **Object(s)**.

This command may be used to read a **Monomorphic-UII** by declaring the relevant **Object-Identifier** as the single entry in the list of **Read-Objects,** combined with **Read-Type** 4. The Data Processor checks that the **Object-Identifier** is registered as part of a **Monomorphic-UII** entry on the ISO/IEC 15961-2 Data constructs register.

— If so, the encoded bytes are de-compacted to the rules defined on the register and included in the response.

— If not, the appropriate error is returned.

The **Read-Objects** command has the following arguments:

Max-App-Length (see 7.4.32)
Read-Objects (see 11.13)
Read-Type (see 7.4.53)
Singulation-Id (see 7.2.1)

Read-Objects command

Module-OID: OBJECT IDENTIFIER = 1 0 15961 126 18

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

Read-Type: INTEGER
Possible Values:

Value	Definition
0	Read-1st-Objects
1	Read-Multiple-Objects
2	Read-All-Objects
3	Read-Monomorphic-UII

Max-App-Length: INTEGER (1..65535)
This only applies to Read-Type (0) and is expressed in bytes

Read-Objects-List: List of <Read-Objects>
This does not apply to Read-All-Objects (2)

If the **Check-Duplicate** argument is set to FALSE in **the Read-Object-List** argument, having found the requested **Object-Identifier** the interrogator shall return the first **Object** found without checking for duplicates. If the **Check-Duplicate** argument is set to TRUE, the interrogator shall check for duplicate **Object-Identifiers**. If more than one instance of the requested **Object-Identifier** is found, the interrogator shall return the first found **Object** and indicate the presence of duplicates with the appropriate **Completion-Code**.

10.11.2 Read-Objects response

The **Read-Objects** response has the following arguments:

Completion-Code (see 9.2)
Execution-Code (see 9.3)
Read-Objects-Response-List (see 11.14)

Read-Objects response

Module-OID: OBJECT IDENTIFIER = 1 0 15961 127 18

Read-Objects-Response-List: List of <Read-Objects-Response>

Completion-Code: INTEGER
Possible Values:

Value	Definition
0	No-Error
8	Singulation-Id-Not-Found
255	Execution-Error

Additional Completion-Codes apply to the Read-Objects-Response-List

Execution-Code: INTEGER
Possible Values: As defined in 9.3.

10.12 Write-Objects-Segmented-Memory-Tag

10.12.1 Write-Objects-Segmented-Memory-Tag command

The **Write-Objects-Segmented-Memory-Tag** command is similar to the **Write-Objects** command except that it is intended to write data to a selected memory bank in a segmented memory tag. The command may be implemented to write initial data to the RFID tag, or to add data to the tag. The command is supported by a compound argument **Add-Objects-List**.

This command shall not be used to write a **Monomorphic-UII**. If the **AFI** on the RFID 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 10.23.

The **Access-Password** in the command is used to match that on the RFID tag to permit writing data to the RFID tag.

The **DSFID-Constructs-list** is used to specify the **Access-Method** and **Data-Format**. The **Ext-DSFID-Constructs-list** is used to set indicators on the **Extended-DSFID** and to instruct the Data Processor to carry out certain procedures, e.g. to apply a CRC to the data.

The **DSFID-Lock** argument, if set, is applied to the entire **DSFID** and **Extended-DSFID** byte string.

The **AFI** and **DSFID** are provided as arguments for use in one of the following ways:

- If data is being written to a blank segmented memory RFID tag, then these arguments are provided as part of this command to minimise communications.
- Depending on the memory bank concerned, if data is being added to the segmented memory RFID tag, then the **AFI** and **DSFID** in the command should match the **AFI** and **DSFID** already encoded on the RFID tag, else there is an error and the encoding process can cease before significant amounts of data have been processed.
- Additionally all the requirements declared by the arguments in the **Ext-DSFID-Constructs-list** shall be processed.

If the **Access-Method** is **Packed-Objects**, all **Objects** specified as arguments shall be included within the same new **Packed-Object** added after any existing **Packed-Objects** in memory. A number of arguments only apply to **Packed-Objects** and these are defined in the **Packed-Object-Constructs** argument.

If the **Access-Method** is **Tag-Data-Profile**, all **Objects** specified as arguments shall be included within the same **Tag-Data-Profile**. Adding an Object that is not specified by the **Tag-Data-Profile-ID-Table** argument shall be treated as an error, and no encoding take place.

Multiple records may be encoded only on Memory Bank 11. Individual records are written only after the MR-header has been created. The details defined in 10.10.1 for Multiple Records shall apply in constructing the command.

The **Write-Objects-Segmented-Memory-Tag** command has the following arguments:

Access-Password (see 7.4.1)
Add-Objects-List (see 11.1)
AFI (see 7.2.2)
DSFID-Constructs (see 11.2)
DSFID-Lock (see 0)
Ext-DSFID-Constructs (see 11.4)
Memory-Bank (see 7.4.33)
Multiple-Records-Constructs (see 11.8)
Packed-Object-Constructs (see 11.12)
Singulation-Id (see 7.2.1)
Tag-Data-Profile-ID-Table (see 7.4.60)

Write-Objects-Segmented-Memory-Tag command

Module-OID: OBJECT IDENTIFIER = 1 0 15961 126 19

Singulation-Id: BYTE STRING (0.255)

Memory-Bank: BIT STRING
Possible Values:

Value	Definition
01	UII-Memory
11	User-Memory

Access-Password: [Optional] BYTE STRING (4 bytes)

AFI: BYTE [only applies if Memory-Bank = 01]

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

Ext-DSFID-Constructs-list: [Optional] List of <Ext-DSFID-Constructs>

DSFID-Lock: BOOLEAN
If set to TRUE, the interrogator shall lock the DSFID

Add-Objects-List: List of <Add-Objects>

Packed-Object-Constructs: [OPTIONAL] List of <Packed-Object-Constructs>

Tag-Data-Profile-ID-Table: INTEGER [OPTIONAL]

Multiple-Records-Constructs: [OPTIONAL] List of <Multiple-Records-Constructs>

10.12.2 Write-Objects-Segmented-Memory-Tag response

The **Write-Objects-Segmented-Memory-Tag** response has the following field names:

Completion-Code (see 9.2)
Execution-Code (see 9.3)
Write-Responses (see 11.18)

Additional **Completion-Codes** apply to each **Object-Identifier** and are incorporated in the **Write-Response** argument.

Write-Objects-Segmented-Memory-Tag response

Module-OID: OBJECT IDENTIFER = 1 0 15961 127 19

Write-Response-List: List of <Write-Responses>

Completion-Code: INTEGER

Possible Values:

Value	Definition
0	No-Error
8	Singulation-Id-Not-Found
25	Password-Mismatch
26	AFI-Mismatch
27	DSFID-Mismatch
33	Insufficient-Tag-Memory
36	Command-Cannot-Process-Monomorphic-UII
37	Data-CRC-Not-Applied
38	Length-Not-Encoded-In-DSFID
43	Data-Format-Not-Compatible-Multiple-Records-Header
44	Access-Method-Not-Compatible-Multiple-Records-Header
45	Sector-Identifier-Not-Compatible-Multiple-Records-Header
46	Record-Preamble-Not-Configured
47	Record-Preamble-Not-Locked
255	Execution Error

Additional Completion-Codes apply to the Write-Response-List

Execution-Code: INTEGER

Possible Values: As defined in 9.3.

10.13 Write-EPC-UII

10.13.1 Write-EPC-UII command

The **Write-EPC-UII** command instructs the interrogator to write an **EPC-Code** into Memory Bank 01 of a segmented memory RFID tag. The command supports various lengths of **EPC-Codes** specified by EPCglobal.

The **Access-Password** in the command is used to match that on the RFID tag to protect against unauthorised writing of data to the RFID tag.

This command can be used to initially write the **EPC-Code** to the RFID tag, or to overwrite the code value. If the new code is of a shorter length, the interrogator needs to ensure that bytes representing part of the older code are removed.

The **Write-EPC-UII-Tag** command has the following arguments:

Access-Password (see 7.4.1)
EPC-Code (see 7.4.19)
Memory-Bank-Lock (see 7.4.34)
NSI-Bits (see 7.4.40)
Singulation-Id (see 7.2.1)

Write-EPC-UII command

Module-OID: OBJECT IDENTIFIER = 1 0 15961 126 20

Singulation-Id: BYTE STRING (0.255)

Access-Password: [Optional] BYTE STRING (4 bytes)

NSI-Bits: BIT STRING

EPC-Code: BYTE STRING

Memory-Bank-Lock: BOOLEAN
If TRUE the entire memory bank shall be locked.

10.13.2 Write-EPC-UII response

The **Write-EPC-UII** response has the following field names:

Completion-Code (see 9.2)
Execution-Code (see 9.3)

Write-EPC-UII response

Module-OID: OBJECT IDENTIFIER = 1 0 15961 127 20

Completion-Code: INTEGER
Possible Values:

Value	Definition
0	No-Error
8	Singulation-Id-Not-Found
25	Password-Mismatch
33	Insufficient-Tag-Memory
255	Execution-Error

Execution-Code: INTEGER
Possible Values: As defined in 9.3.

10.14 Inventory-ISO-UIImemory

10.14.1 Inventory-ISO-UIImemory command

The **Inventory-ISO-UIImemory** command is intended to return the contents of the UII memory from a number of segmented memory tags, given the expectation that an **Object-Identifier** for a UII other than an **EPC-Code** is encoded. The response returns the content of the UII memory for all tags whose encoded bit string matches the arguments of the command.

The arguments provided in the command enable a bit mask to be incorporated into appropriate air interface protocol commands to select only tags that match the bit mask. The **AFI** is a required argument, and the other two arguments extend the bit string to increase the capability of selection.

This command may be used to read a **Monomorphic-UII** by declaring the relevant **AFI** as an argument. The Data Processor checks that the **AFI** is registered as part of a **Monomorphic-UII** entry on the ISO/IEC 15961-2 Data constructs register.

— If so, the encoded bytes are de-compacted to the rules defined on the register and included in the response.

— If not, the appropriate error is returned

NOTE A **DSFID** is not required when invoking this command to inventory Monomorphic-UIIs.

The **Inventory-ISO-UIImemory** command has the following arguments:

Additional-App-Bits (see 7.4.2)
AFI (see 7.2.2)
DSFID-Constructs (see 11.2)

Inventory-ISO-UIImemory command

Module-OID: OBJECT IDENTIFIER = 1 0 15961 126 21

AFI: BYTE

DSFID-Constructs-list: [OPTIONAL] List of <DSFID-Constructs>

Additional-App-Bits: BIT STRING [Optional]

10.14.2 Inventory-ISO-UIImemory response

The **Inventory-ISO-UIImemory** response has the following field names:

Completion-Code (see 9.2)
Execution-Code (see 9.3)
ISO-UIImemory (see 11.5)

Inventory-ISO-UIImemory response

Module-OID: OBJECT IDENTIFIER = 1 0 15961 127 21

ISO-UII Memory-List: List of <ISO-UIIMemory>

Completion-Code: INTEGER
Possible Values:

Value	Definition
0	No-Error
255	Execution-Error

Execution-Code: INTEGER
Possible Values: As defined in 9.3.

10.15 Inventory-EPC-UIImemory

10.15.1 Inventory-EPC-UIImemory command

The **Inventory-EPC-UIImemory** command is intended to return the contents of the UII memory from a number of segmented memory tags, given the expectation that an **EPC-Code** is encoded. The response returns the content of the UII memory for all tags whose encoded bit string matches the arguments of the command.

The arguments provided in the command enable a bit mask to be incorporated into appropriate air interface protocol commands to select only tags that match the bit mask. The **Tag-Mask** value consists of a bit string that should be determined by reference to relevant EPCglobal standards. The same applies to the **Pointer**, which identifies the first bit of a continuous string on the RFID tag that needs to be matched with the **Tag-Mask**.

The **Inventory-EPC-UIImemory** command has the following arguments:

Length-Of-Mask (see 7.4.27)
Pointer (see 7.4.50)
Tag-Mask (see 7.4.61)

Inventory-EPC-UIImemory command

Module-OID: OBJECT IDENTIFIER = 1 0 15961 126 22

Pointer: HEXADECIMAL ADDRESS
The address of the first (msb) bit against which to apply the Tag-Mask.

Length-Of-Mask: INTEGER

Tag-Mask: BIT STRING

10.15.2 Inventory-EPC-UIImemory response

The **Inventory-EPC-UIImemory** response has the following field names:

Completion-Code (see 9.2)
EPC-UIImemory (see 11.3)
Execution-Code (see 9.3)

Inventory-EPC-UIImemory response

Module-OID: OBJECT IDENTIFIER = 1 0 15961 127 22

EPC-UIImemory-List: List of <EPC-UIImemory>

Completion Code: INTEGER
Possible Values:

Value	Definition
0	No-Error
255	Execution-Error

Execution-Code: INTEGER
Possible Values: As defined in 9.3.

10.16 Write-Password-Segmented-Memory-Tag

10.16.1 Write-Password-Segmented-Memory-Tag command

The **Write-Password-Segmented-Memory-Tag** command instructs the interrogator to write one of the **Passwords** defined in the command to the appropriate memory of a segmented memory RFID tag. Only one **Password** may be specified per command. The **Password-Type** identifies to the Data Processor the type of **Password**, and therefore its storage location on the RFID tag.

The **Write-Password-Segmented-Memory-Tag** command has the following arguments:

Password (see 7.4.46)
Password-Type (see 7.4.47)
Singulation-Id (see 7.2.1)

Write-Password-Segmented-Memory-Tag command

Module-OID: OBJECT IDENTIFIER = 1 0 15961 126 23

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

Password-Type: INTEGER
Possible Values:

Value	Definition
0	Kill-Password
1	Access-Password

Password: BYTE STRING
For **Password-Types** 0 and 1, the length is 4 bytes

10.16.2 Write-Password-Segmented-Memory-Tag response

The **Write-Password-Segmented-Memory-Tag** response has the following field names:

Completion-Code (see 9.2)
Execution-Code (see 9.3)

Write-Password-Segmented-Memory-Tag response

Module-OID: OBJECT IDENTIFIER = 1 – 15961 127 23

Completion-Code: INTEGER
Possible Values:

Value	Definition
0	No-Error
8	Singulation-Id-Not-Found
26	Password-Not-Written
33	Insufficient-Tag-Memory
255	Execution-Error

Execution-Code: INTEGER
Possible Values: As defined in 9.3.

10.17 Read-Words-Segmented-Memory-Tag

10.17.1 Read-Words-Segmented-Memory-Tag command

The **Read-Words-Segmented-Memory-Tag** command instructs the interrogator to read a contiguous sequence of words from one of the memory banks of a segmented memory RFID tag. This command can be used to extract encoded bytes, which might not be Object-based such as the unique **Singulation-Id** or a password. It can also be useful for diagnostic purposes.

The **Read-Words-Segmented-Memory-Tag** command has the following arguments:

Access-Password (see 7.4.1)
Memory-Bank (see 7.4.33)
Singulation-Id (see 7.2.1)
Word-Count (see 7.4.63)
Word-Pointer (see 7.4.64)

Read-Words-Segmented-Memory-Tag command

Module-OID: OBJECT IDENTIFIER = 1 0 15961 126 24

Singulation-Id: BYTE STRING (0.255)

Memory-Bank: BIT STRING
Possible Values: (00..11)

Word-Pointer: HEXADECIMAL ADDRESS

Word-Count: INTEGER

Access-Password: [Optional] BYTE STRING (4 bytes)

10.17.2 Read-Words-Segmented-Memory-Tag response

The **Read-Words-Segmented-Memory-Tag** response has the following field names:

Completion-Code (see 9.2)
Execution-Code (see 9.3)
Read-Data (see 7.5.12)

Read-Words-Segmented-Memory-Tag response

Module-OID: OBJECT IDENTIFIER = 1 0 15961 127 24

Read-Data: BYTE STRING

Completion-Code: INTEGER
Possible Values:

Value	Definition
0	No-Error
25	Password-Mismatch
254	Undefined-Command-Error
255	Execution-Error

Execution-Code: INTEGER
Possible Values: As defined in 9.3.

10.18 Kill-Segmented-Memory-Tag

10.18.1 Kill-Segmented-Memory-Tag command

The **Kill-Segmented-Memory-Tag** command instructs the interrogator to apply appropriate air interface protocols to render the RFID tag unreadable in future. The **Kill-Password** in the command must match the Password encoded on the RFID tag.

The **Kill-Segmented-Memory-Tag** command has the following arguments:

Kill-Password (see 7.4.26)
Singulation-Id (see 7.2.1)

Kill-Segmented-Memory-Tag command

Module-OID: OBJECT IDENTIFIER = 1 0 15961 126 25

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

Kill-Password: BYTE STRING (4 bytes)

10.18.2 Kill-Segmented-Memory-Tag response

The **Kill-Segmented-Memory-Tag** response has the following field names:

Completion-Code (see 9.2)
Execution-Code (see 9.3)

Kill-Segmented-Memory-Tag response

Module-OID: OBJECT IDENTIFIER = 1 0 15961 127 25

Completion-Code: INTEGER
Possible Values:

Value	Definition
0	No-Error
8	Singulation-Id-Not-Found
27	Zero-Kill-Password-Error
28	Kill-Failed
255	Execution-Error

Execution-Code: INTEGER
Possible Values: As defined in 9.3.

10.19 Delete-Packed-Object

10.19.1 Delete-Packed-Object command

The **Delete-Packed-Object** command instructs the interrogator to delete the **Packed-Object** in which the specified **Object-Identifier** is contained. The **Object-Identifier** simply acts as an alias to identify a specific **Packed-Object**. Only one RFID 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 associated **Packed-Object** from the tag and replacing this with pad bytes.

This procedure might not succeed if the **Packed-Object** is locked, if this is found to be the case, the response will return the appropriate **Completion-Code**.

If the **Check-Duplicate** flag is set to TRUE, the interrogator shall verify, before deleting the requested **Packed-Object**, that there is only a single instance of the requested **Object-Identifier** on the RFID tag. If the interrogator detects that the RFID tag is encoding more than one instance of the referenced **Object-Identifier**, it shall not perform the **Delete-Packed-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 **Packed-Object** that contains the specified by its **Object-Identifier**.

NOTE 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-Packed-Object** command has the following arguments:

Check-Duplicate (see 7.4.10)
Object-Identifier (see7.3.2)
Singulation-Id (see 7.2.1)

Delete-Packed-Object command

Module-OID: OBJECT IDENTIFIER = 1 0 15961 126 26

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 Object-Identifier on the RFID tag

10.19.2 Delete-Packed-Object response

The **Delete-Packed-Object** response has the following field names:

Completion-Code (see 9.2)
Execution-Code (see 9.3)

Delete-Packed-Object response

Module-OID: OBJECT IDENTIFIER = 1 0 15961 127 26

Completion-Code: INTEGER

Possible Values:

Value	Definition
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

Possible Values: As defined in 9.3.

10.20 Modify-Packed-Object-Structure

10.20.1 Modify-Packed-Object-Structure command

The **Modify-Packed-Object-Structure** command is used to change the structure of a **Packed-Object**. The structure of a **Packed-Object** may be modified to define a specific directory type if the **Packed-Object** was created with the **Packed-Object** pointer allocation type as specified in 7.4.48. The command determines which type of directory shall be applied. The **Object-Identifier** simply acts as an alias to identify a specific **Packed-Object**. Only one RFID tag and only one **Object-Identifier** shall be programmed per command to ensure that the modification process is robust.

This procedure might not succeed if the **Packed-Object** is locked, if this is found to be the case, the response will return the appropriate **Completion-Code**.

A number of arguments are defined in the **Packed-Object-Constructs** argument. These arguments shall be used if included, but only if they are relevant for the type of Packed Object being modified by the command. If they are not relevant for the current Packed Object, the parameters shall be ignored.

If the **Packed-Object** already has a specific directory type defined, then the appropriate **Completion-Code** will be returned.

If the **Check-Duplicate** flag is set to TRUE, the interrogator shall verify, before modifying the requested **Packed-Object**, that there is only a single instance of the requested **Object-Identifier** on the RFID tag. If the interrogator detects that the RFID tag is encoding more than one instance of the referenced **Object-Identifier**, it shall not perform the **Modify-Packed-Object-Structure** function and shall return the appropriate **Completion-Code**.

If the **Check-Duplicate** flag is set to FALSE, the interrogator shall modify the first occurrence of the **Packed-Object** that contains the specified by its **Object-Identifier**.

NOTE 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 **Modify-Packed-Object-Structure** command has the following arguments:

Check-Duplicate (see 7.4.10)
Object-Identifier (see7.3.2)
Packed-Object-Constructs (see 11.12)
Singulation-Id (see 7.2.1)

Modify-Packed-Object command

Module-OID: OBJECT IDENTIFIER = 1 0 15961 126 27

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 Object-Identifier

Packed-Object-Constructs-List: [OPTIONAL] List of <Packed-Object-Constructs>

10.20.2 Modify-Packed-Object-Structure response

The **Modify-Packed-Object** response has the following field names:

Completion-Code (see 9.2)
Execution-Code (see 9.3)

Modify-Packed-Object-Structure response

Module-OID: OBJECT IDENTIFIER = 1 0 15961 127 27

Completion-Code: INTEGER
Possible Values:

Value	Definition
0	No-Error
7	Object-Locked-Could-Not-Modify
8	Singulation-Id-Not-Found
10	Duplicate-Object
13	Object-Identifier-Not-Found
30	Directory-Already-Defined
33	Insufficient-Tag-Memory
37	Data-CRC-Not-Applied
38	Length-Not-Encoded-In-DSFID
255	Execution-Error

Execution-Code: INTEGER
Possible Values: As defined in 9.3.

10.21 Write-Segments-6TypeD-Tag

10.21.1 Write-Segments-6TypeD-Tag command

The **Write-Segments-6TypeD-Tag** command instructs the ISO/IEC 15962 Data Processor to write data to the UII segment, the item-related segment or both segments. The command may be implemented to write initial data to the RFID tag, or to add data to the tag. The command is supported by a compound argument **Add-Objects-List**, which applies to the Item-related segment.

This command shall not be used to write a **Monomorphic-UII**. If the **AFI** on the RFID 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 10.23.

If data is being encoded on the RFID tag for the first time, the **AFI**, **UII-DSFID**, and **Item-Related-DSFID** (if that segment is being encoded) shall be incorporated in the encoded byte string. If any of these three is already encoded on the tag and there is a mis-match with the equivalent argument in the command, then the procedure shall be aborted

If the **Access-Method** is **Packed-Objects**, all **Objects** specified as arguments shall be included within the same new **Packed-Object** added after any existing **Packed-Objects** in memory. A number of arguments only apply to **Packed-Objects** and these are defined in the **Packed-Object-Constructs** argument.

If the **Access-Method** is **Tag-Data-Profile**, all **Objects** specified as arguments shall be included within the same **Tag-Data-Profile**. Adding an Object that is not specified by the **Tag-Data-Profile-ID-Table** argument shall be treated as an error, and no encoding take place.

Multiple records may be encoded only on the item-related segment. Individual records are written only after the MR-header has been created. The details defined in 10.10.1 shall apply in constructing the command.

The **Write-Segments-6TypeD-Tag** command has the following arguments:

AFI (see 7.2.2)
Item-Related-Add-Objects-List (see 11.6)
Item-Related-DSFID-Constructs (see 11.7)
Lock-UII-Segment-Arguments (see 7.4.31)
Memory-Segment (see 7.4.36)
Multiple-Records-Constructs (see 11.8)
Packed-Object-Constructs (see 11.12)
Singulation-Id (see 7.2.1)
Tag-Data-Profile-ID-Table (see 7.4.60)
UII-Add-Objects-List (see 11.16)
UII-DSFID-Constructs (see 11.17)

Write-Segments-6TypeD-Tag command

Module-OID: OBJECT IDENTIFIER = 1 0 15961 126 28

Singulation-Id: BYTE STRING (0.255)

Memory-Segment: INTEGER

Possible Values:

Value	Definition
1	UII segment
2	Item-related data segment
3	Both segments presented as objects

AFI: BYTE

Not required for **Memory-Segment** =2 (Item-related data segment)

UII-DSFID-Constructs-list: [OPTIONAL] List of <DSFID-Constructs>

Not required for Memory-Segment = 2 (Item-related data segment)

UII-Add-Objects-List: List of <Add-Objects>

Only required for Memory-Segment = 1 (UII segment), and this shall only contain one **Object-Identifier** and **Object**

Lock-UII-Segment-Arguments: INTEGER

This argument takes precedence over lock argument elsewhere in the command.

Possible Values:

Value	Definition
1	Protocol Control word
2	DSFID
3	PC word + DSFID
4	UII Data-Set, but not PC word nor DSFID
5	PC word + UII Data-Set (this combination can be applied to a Monomorphic-UII)
6	DSFID + UII Data-Set
7	The complete UII segment, including the CRC-16

Item-Related-DSFID-Constructs-list: [OPTIONAL] List of <DSFID-Constructs>

Not required for Memory-Segment = 1 (UII segment)

Item-Related-Add-Objects-List: List of <Add-Objects>

Not required for Memory-Segment = 1 (UII segment)

Packed-Object-Constructs: [OPTIONAL] List of <Packed-Object-Constructs>

Not required for Memory-Segment = 1 (UII segment)

Tag-Data-Profile-ID-Table: INTEGER [OPTIONAL]

Not required for Memory-Segment = 1 (UII segment)

Multiple-Records-Constructs: [OPTIONAL] List of <Multiple-Records-Constructs>

10.21.2 Write-Segments-6TypeD-Tag response

The **Write-Segments-6TypeD-Tag** response has the following field names:

Completion-Code (see 9.2)
Execution-Code (see 9.3)

Write-Segments-6TypeD-Tag response

Module-OID: OBJECT IDENTIFIER = 1 – 15961 127 28

Completion-Code: INTEGER

Possible Values:

Value	Definition
0	No-Error
8	Singulation-Id-Not-Found
26	AFI-Mismatch
27	DSFID-Mismatch
31	Packed-Object-ID-Table-Not-Recognised-No-Encoding
32	Tag-Data-Profile-ID-Table-Not-Recognised
33	Insufficient-Tag-Memory
36	Command-Cannot-Process-Monomorphic-UII
43	Data-Format-Not-Compatible-Multiple-Records-Header
44	Access-Method-Not-Compatible-Multiple-Records-Header
45	Sector-Identifier-Not-Compatible-Multiple-Records-Header
46	Record-Preamble-Not-Configured
47	Record-Preamble-Not-Locked
255	Execution-Error

Additional Completion-Codes apply to the Write-Response-List

Execution-Code: INTEGER

Possible Values: As defined in 9.3.

10.22 Read-Segments-6TypeD-Tag

10.22.1 Read-Segments-6TypeD-Tag command

This command shall not be used to read any data object or other part of a multiple record. The relevant procedures are defined in 10.26.

The **Read-Segments-6TypeD-Tag** command instructs the interrogator to read a contiguous sequence of words from all the memory segments of an ISO/IEC 18000-6 Type D RFID tag, and then to subdivide into the segments and other component parts. This command can also be useful for diagnostic purposes.

As the air interface protocol is capable of delivering the entire payload of the Type D tag, it is possible for the application to call for **Object-Identifier(s)** and **Object(s)** from either the UII segment and /or the Item-related data segment. A basic assumption of this command is that only one **Object-Identifier** and **Object** data set is encoded in the UII memory.

This command may be used to read a **Monomorphic-UII** by declaring the relevant **AFI** as an argument. The Data Processor checks that the **AFI** is registered as part of a **Monomorphic-UII** entry on the ISO/IEC 15961-2 Data constructs register.

— If so, the encoded bytes are de-compacted to the rules defined on the register and included in the response.

— If not, the appropriate error is returned

NOTE A **UII-DSFID** is not required when invoking this command read a Monomorphic-UII.

The argument **Read-Type** is only applied if **Objects** are to be returned from the item-related segment. The argument enables the application to request selected and declared **Object-Identifier(s)** to be processed or to request that all **Object-Identifier(s)** be processed.

It is possible, setting **Memory-Segment** = 4 to call for all the bytes as encoded on the UII segment and the item-related segment, in effect the byte string defined in ISO/IEC 18000-6 as the output of the interrogator. As this by-passes any decode process by the ISO/IEC 15962 Data Processor the byte string in the response can be used for diagnostic purposes on the original encoding.

The **Read-Segments-6TypeD-Tag** command has the following arguments:

AFI (see 7.2.2)
Item-Related-DSFID-Constructs (see 11.7)
Memory-Segment (see 7.4.36)
Read-Objects (see 11.13)
Read-Type (see 7.4.53)
Singulation-Id (see 7.2.1)
UII-DSFID-Constructs (see 11.17)

Read-Segments-6TypeD-Tag command

Module-OID: OBJECT IDENTIFIER = 1 0 15961 126 29

Singulation-Id: BYTE STRING (0.255)

Memory-Segment: INTEGER
Possible Values:

Value	Definition
1	UII segment
2	Item-related data segment
3	Both segments presented as objects
4	Both segments as un-interpreted bytes

AFI: BYTE

UII-DSFID-Constructs-list: [OPTIONAL] List of <DSFID-Constructs>
Not required for Memory-Segment = 2 (Item-related data segment)
Not required if the AFI is for a monomorphic-UII

Item-Related-DSFID-Constructs-list: [OPTIONAL] List of <DSFID-Constructs>
Not required for Memory-Segment = 1 (UII segment)

Read-Type: INTEGER
Not required for Memory-Segment = 1 (UII segment)
Possible Values:

Value	Definition
1	Read-Multiple-Objects
2	Read-All-Objects

Read-Objects-List: List of <Read-Objects>
This does not apply to Read-All-Objects (2)

10.22.2 Read-Segments-6TypeD-Tag response

The **Read-Segments-6TypeD-Tag** response has the following arguments:

Completion-Code (see 9.2)
Execution-Code (see 9.3)
ISO-UIImemory (see 11.5)
Length-Lock Byte (see 7.5.3)
Read-Data (see 7.5.12)
Read-Objects-Response-List (see 11.14)

Read-Segments-6TypeD-Tag response

Module-OID: OBJECT IDENTIFIER = 1 0 15961 127 29

ISO-UII Memory-List: List of <ISO-UIIMemory>
This response argument is included when the content of the UII segment has been requested

Length-Lock Byte: BYTE
This response argument is included when the content of the item-related data segment has been requested

Read-Objects-Response-List: List of <Read-Objects-Response>
This response argument is included when the content of the item-related data segment has been requested

Read-Data: BYTE STRING
This response argument is included when the Memory-Segment = 4 (Both segments as un-interpreted bytes)

Completion Code: INTEGER
Possible Values:

Value	Definition
0	No-Error
8	Singulation-Id-Not-Found
255	Execution-Error

Execution-Code: INTEGER
Possible Values: As defined in 9.3.

10.23 Write-Monomorphic-UII

10.23.1 Write-Monomorphic-UII command

The **Write-Monomorphic-UII** command instructs the ISO/IEC 15962 Data Processor to write a **Monomorphic-UII**, either initially or to modify an existing **Monomorphic-UII**. Arguments are applied selectively depending on the type of RFID tag being addressed.

A **Monomorphic-UII** is only supported by an encoded **AFI**, and no **DSFID** is encoded. The **Object-Identifier** is only required for communication purposes between the application and the Data Processor. It is not encoded on the RFID tag. Only **Compact-Parameter** with the value 5 (Monomorphic-UII) shall be used in the command.

The generic encoding process calls for the Data Processor to check that the **AFI** in the command matches with an one for a **Monomorphic-UII** on the ISO/IEC 15961-2 Data Constructs register. If a match is not possible either because the **AFI** is not registered for a **Monomorphic-UII** or that no matching **AFI** can be found on the register then the encoding process is aborted.

If the **AFI** matches, the **Object-Identifier** is also checked for a match with that on the ISO/IEC 15961-2 Data Constructs register. A mismatch generates an appropriate **Completion-Code**, but this should only treated as a warning about constructing the command. The process continues because the **Object-Identifier** is not encoded. The Data Processor uses the explicitly defined compaction scheme associated with the **AFI** on the ISO/IEC 15961-2 Data Constructs register to encode the **Monomorphic-UII**.

If the **Memory-Type** is 0 (18000-6 Type C MB01) the first requirement is to read the content of MB 01 to establish any existing encoding. If this is locked the process is aborted, otherwise the process continues. The encoding process compacts the **Object,** and if this results in an odd number of bytes appends the terminator byte 00_{16}. The Protocol Control word is constructed, incorporating the **AFI** and the length bits. If the **Access-Password** is in the command it is used to match that on the RFID tag to protect against unauthorised writing of data to the RFID tag. If this command is used to over-write MB01 with a new **Monomorphic-UII** it is necessary to compare the length of the current and the new byte string. If the new code is of a shorter length, the interrogator needs to ensure that bytes representing part of the older code are over-written with zero bytes.

If the **Memory-Type** is 1 (18000-6 Type D UII segment) the first requirement is to read the content of the UII segment to establish any existing encoding. If this is locked the process is aborted, otherwise the process continues. The encoding process compacts the **Object,** and if this results in an odd number of bytes appends the terminator byte 00_{16}. The Protocol Control word is constructed, incorporating the **AFI** and the length bits. If this command is used to over-write the UII segment with a new **Monomorphic-UII** it is necessary to compare the length of the current and the new byte string. If the new code is of a different length (i.e. shorter or longer) to an existing **Monomorphic-UII**, the interrogator needs to check if an item related segment is already encoded. The over-writing procedure, as defined in ISO/IEC 15962, needs to take into account whether there is item-related data on the RFID and whether any of this is locked. This is necessary, because encoding on an ISO/IEC 18000-6 Type D tag is contiguous between segments.

If the **Memory-Type** is 2 (single memory tag) the first requirement is to read the AFI on the RFID tag (which can be encoded in a separate memory area) and at least 16 bytes from the content of the user memory to establish any existing encoding. If encoded bytes are found, then the process continues until a string of four zero bytes is found. If either the **AFI** or any part of the user memory is locked the process is aborted, otherwise the process continues. The encoding process compacts the **Object,** and adds the length of the compacted **Monomorphic-UII** as a prefix. If this command is used to over-write with a new **Monomorphic-UII** it is necessary to compare the length of the current and the new byte string. If the new code is of a shorter length, the interrogator needs to ensure that bytes representing part of the older code are over-written with zero bytes. If the correct **AFI** for the **Monomorphic-UII** is not already encoded on the RFID tag, then this is encoded.

The **Write-Monomorphic-UII** command has the following arguments:

Access-Password (see 7.4.1)
AFI (see 7.2.2)
AFI-Lock (see 7.4.3)
Compact-Parameter (see 7.3.5)
Memory-Bank-Lock (see 7.4.34)
Memory-Type (see 7.4.37)
Object (see 7.3.4)
Object-Identifier (see 7.3.2)
Object-Lock (see 7.3.6)
Singulation-Id (see 7.2.1)

Write-Monomorphic-UII command

Module-OID: OBJECT IDENTIFIER = 1 0 15961 126 30

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

Memory-Type: INTEGER (0..15)

Possible Values:

Value	Definition
0	18000-6 Type C MB01
1	18000-6 Type D UII segment
2	single memory tag

Access-Password: [OPTIONAL] BYTE STRING (4 bytes)
This may only be applied for **Memory-Type** 0 (18000-6 Type C MB01). Additionally it is optional.

AFI: BYTE

AFI-Lock: BOOLEAN
This may only be applied for **Memory-Type** 2 (single memory tag)
If set to TRUE, the interrogator shall lock the AFI

Object-Identifier: OBJECT IDENTIFIER
This is the **Object-Identifier** associated with the AFI on the ISO/IEC 15961 Data Constructs register.

Object: BYTE STRING
This is the Monomorphic-UII as presented by the application

Compact-Parameter: INTEGER
The only permitted value is 5 (Monomorphic-UII), which shall cause the Data Processor to call the explicitly defined compaction scheme associated with the AFI on the ISO/IEC 15961 Data Constructs register.

Object-Lock: BOOLEAN
This may only be applied for **Memory-Type** 2 (single memory tag)
If TRUE the interrogator shall lock the relevant Data-Set

Memory-Bank-Lock: BOOLEAN
This may only be applied for **Memory-Type** 0 (18000-6 Type C MB01)
If TRUE the entire memory bank shall be locked.

Lock-UII-Segment-Arguments: INTEGER
This may only be applied for **Memory-Type** 1 (18000-6 Type D UII segment)
Possible Values:

Value	Definition
4	UII Data-Set, but not PC word nor DSFID
5	PC word + UII Data-Set (this combination can be applied to a Monomorphic-UII)
7	The complete UII segment, including the CRC-16

10.23.2 Write-Monomorphic-UII response

The **Write-Monomorphic-UII** response has the following field names:

Completion-Code (see 9.2)
Execution-Code (see 9.3)

Write-Monomorphic-UII response

Module-OID: OBJECT IDENTIFIER = 1 0 15961 127 30

Completion-Code: INTEGER

Possible Values:

Value	Definition
0	No-Error
7	Object-Locked-Could-Not-Modify
8	Singulation-Id-Not-Found
22	Object-Modified-But-Not-Locked
25	Password-Mismatch
26	AFI-Mismatch
33	Insufficient-Tag-Memory
34	AFI-Not-For-Monomorphic-UII
35	Monomorphic-UII-OID-Mismatch
255	Execution-Error

Execution-Code: INTEGER

Possible Values: As defined in 9.3.

10.24 Configure-Extended-DSFID

The **Configure-DSFID** command (see 10.2) is structure to support the encoding of an **Access-Method** and a **Data-Format.** Since the publication of ISO/IEC 15961:2004, additional features have been added to create an Extended DSFID, for example, to signal the length of encoded data or whether error detection has been added to the data for data that is stored for a long period of time. The **Configure-Extended-DSFID** command is used to set various indicators that are encoded within the Extended DSFID. As the **DSFID** specifies which encoding rule to follow the additional features declared by this command simply determine additional processes that are to be used when encoding data.

10.24.1 Configure-Extended-DSFID command

The **Configure-Extended-DSFID** command is used to instruct the Data Processor to encode all relevant DSFID features on the RFID tag. The command needs to be invoked before any encoding takes place on the RFID tag.

The **DSFID-Constructs-list** is used to specify the **Access-Method** and **Data-Format**. The **Ext-DSFID-Constructs-list** is used to set indicators on the **Extended-DSFID** and to instruct the Data Processor to carry out certain procedures, e.g. to apply a CRC to the data.

The **DSFID-Lock** argument, if set, is applied to the entire **DSFID** and **Extended-DSFID** byte string.

The **Configure-DSFID** command has the following arguments:

DSFID-Constructs (see 11.2)
DSFID-Lock (see 0)
Ext-DSFID-Constructs (see 11.4)
Singulation-Id (see 7.2.1)

The **DSFID-Lock** argument should only be applied once the user decides that all the features of the Extended DSFID can be permanently encoded.

Configure-Extended-DSFID command

Module-OID: OBJECT IDENTIFIER = 1 0 15961 126 31

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

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

Ext-DSFID-Constructs-list: [Optional] List of <Ext-DSFID-Constructs>

DSFID-Lock: BOOLEAN
If set to TRUE, the interrogator shall lock the DSFID

10.24.2 Configure-Extended-DSFID response

The **Configure-Extended-DSFID** response has the following field names:

Completion-Code (see 9.2)
Execution-Code (see 9.3)

Configure-Extended-DSFID response

Module-OID: OBJECT IDENTIFIER = 1 0 15961 127 31

Completion-Code: INTEGER
Possible Values:

Value	Definition
0	No-Error
4	DSFID-Not-Configured
5	DSFID-Not-Configured-Locked
6	DSFID-Configured-Lock-Failed
8	Singulation-Id-Not-Found
37	Data-CRC-Not-Applied
38	Length-Not-Encoded-In-DSFID
255	Execution-Error

Execution-Code: INTEGER
Possible Values: As defined in 9.3.

10.25 Configure-Multiple-Records-Header

The **Configure-Multiple-Records-Header** command shall be the first command invoked to enable an RFID tag to support multiple records. The command can be applied to any tag of sufficient memory capacity, but only applies to Memory Bank 11 of a segmented memory tag such as ISO/IEC 18000-6 Type C.

10.25.1 Configure-Multiple-Records-Header command

The **Configure-Multiple-Records-Header** command is used to instruct the Data Processor to encode all relevant features of the MR-header on the RFID tag. The command needs to be invoked before any encoding of a record takes place on the RFID tag.

The **Access-Password** in the command is used to match that on the RFID tag to permit writing data to the RFID tag. This only applies to the ISO/IEC 18000-6 Type C tag, and can only be applied to that RFID tag if such a password had previously been encoded.

The **DSFID-Constructs-list** is used to specify the **Access-Method** (= 4) and **Data-Format**. If all the records are known to share the same **Data-Format**, defined as homogenous **Multiple-Records** encoding, then the value of that **Data-Format** shall be encoded. If any of the records are expected to have different **Data-Formats**, defined as a heterogeneous **Multiple-Records** encoding, then the **Data-Format** {00001} shall be used.

NOTE For the **Multiple-Records Access-Method**, the interpretation of **Data-Format** {00001} is that the records do not share a common **Data-Format**.

Because **Access-Method** {4} is required, the **Ext-DSFID-Constructs-list** argument (see 11.4) is required to be used, but with these specific variants:

— **Memory-Length-Encoding** is optional and applies to the directory. The possible values for the bit string are 00 to indicate that no encoding is requested, and 10 to indicate that the current length of the directory shall be calculated by the Data Processor and encoded and encoded in the MR-header. If the MR-header is locked, then this cannot be used.

— **Data-CRC-Indicator** is optional and is applied to the entire directory component, but is recommended if a directory is present. The only possible value for the bit string is 10.

— **Simple-Sensor-Indicator** is conditional and only required if other mechanisms are not available on the RFID tag to declare the presence of simple sensors.

— **Battery-Assist-Indicator** is conditional and only required if other mechanisms are not available on the RFID tag to declare the presence of a battery.

— **Full-Function-Sensor-Indicator** is conditional and only required if other mechanisms are not available on the RFID tag to declare the presence of full function sensors.

— **DSFID-Pad-Bytes** shall not be used because of the facility to add pad bytes for the complete MR-header.

The **Directory-Length-EBV8-Indicator** (see 7.4.14) is mandatory and used to instruct the Data Processor to provide sufficient space in the MR-header to allow for the directory size to be encoded within the size limits set by the application.

The **Multiple-Records-Feature-Indicator** is mandatory and is a bit mapping to provide further information about the records encoded in the Logical Memory (see 7.4.39 for the details).

The **Sector-Identifier** (see 7.4.57) when included in this argument as a non-zero value represents the true sector identifier, but a zero value indicates that the true sector identifier is encoded in each of the individual records.

The **Pointer-To-Multiple-Records-Directory** (see 7.4.51) is an EBV-8 address to shows the start of the directory. The specific values are derived from air interface responses that provide details of the hardware memory mapping of the RFID tag.

The **Lock-Multiple-Records-Header** argument (see 7.4.29) is used to determine whether the entire MR-header is locked or not. If it is not locked the following options are possible:

— to leave only the data length of the directory field unlocked

— to leave only the number of records field unlocked

— to leave both the data length of the directory field and the number of records field unlocked

In the unlocked cases the Data Processor shall ensure that necessary pad bytes are added after the preceding fields to achieve block alignment before they are locked. It shall also block align the unlocked field(s).

The **Configure-Multiple-Records-Header** command has the following arguments:

Access-Password (see 7.4.1)
Directory-Length-EBV8-Indicator (see 7.4.14)
DSFID-Constructs (see 11.2)
Ext-DSFID-Constructs (see 11.4)
Lock-Multiple-Records-Header (see 7.4.29)
Multiple-Records-Features-Indicator (see 7.4.39)
Pointer-To-Multiple-Records-Directory (see 7.4.51)
Sector-Identifier (see 7.4.57)
Singulation-Id (see 7.2.1)

Configure-Multiple-Records-Header command

Module-OID: OBJECT IDENTIFIER = 1 0 15961 126 32

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

Access-Password: [OPTIONAL] BYTE STRING (4 bytes)

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

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

Directory-Length-EBV8-Indicator: [Optional] INTEGER
Possible Values:

Value	Definition
1	Single byte EBV-8
2	Double byte EBV-8, even if the length is less than 128 blocks

Multiple-Records-Features-Indicator: BYTE
This is a bit map that is set in this command that determines rules for the Data Processor to follow when encoding individual records.

Sector-Identifier: INTEGER
Possible Values:

Value	Definition
0	Indicates that the sector identifier varies between records, and that the true value is only obtainable for the individual record.
≠0	Indicates the true value of the sector that applies to all records.

Pointer-To-Multiple-Records-Directory: EBV-8 VALUE
This value is based on responses from the interrogator on the memory mapping of the RFID tag that is being addressed.

Lock-Multiple-Records-Header: INTEGER
Possible Values:

Value	Definition
0	Not locked
1	Completely locked
2	All components locked except for the Number of records field.

10.25.2 Configure-Multiple-Records-Header response

The **Configure-Multiple-Records-Header** response has the following field names:

Completion-Code (see 9.2)
Execution-Code (see 9.3)

Configure-Multiple-Records-Header response

Module-OID: OBJECT IDENTIFIER = 1 0 15961 127 32

Completion-Code: INTEGER
Possible Values:

Value	Definition
0	No-Error
4	DSFID-Not-Configured
8	Singulation-Id-Not-Found
25	Password-Mismatch
37	Data-CRC-Not-Applied
38	Length-Not-Encoded-In-DSFID
39	Multiple-Records-Header-Not-Configured
40	Multiple-Records-Header-Not-Locked
41	File-Support-Indicators-Not-Configured
42	File-Support-Indicators-Not-Locked
255	Execution-Error

Execution-Code: INTEGER
Possible Values: As defined in 9.3.

10.26 Read-Multiple-Records

10.26.1 Read-Multiple-Records command

The **Read-Multiple-Records** command instructs the interrogator to read various logically structured components from an RFID tag configured to encode multiple records. This includes reading a set of one or more **Object-Identifiers** and associated **Objects** from an individual record. Only one RFID tag shall be programmed per command to ensure that the reading process is robust.

The command is applied to memory bank 11 of the ISO/IEC 18000-6 Type C tag, and to the Item-related data segment of the ISO/IEC 18000-6 Type D tag.

The **Read-Record-Type** argument (see 7.4.52) provides a set of options for read data from the tag.

The **Read-Multiple-Records** command has the following arguments:

Max-App-Length (see 7.4.32)
Read-Objects (see 11.13)
Read-Record-Type (see 7.4.52)
Singulation-Id (see 7.2.1)

Read-Multiple-Records command

Module-OID: OBJECT IDENTIFIER = 1 0 15961 126 33

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

Read-Record-Type: INTEGER

Possible Values:

Value	Definition
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

Read-Objects-List: List of <Read-Objects>

This only applies to Read-Preamble-Specific-Multiple-Record (3), Read-All-Record-OIDs-Specific-Record-Type (4), Read-All-Objects-Specific-Multiple-Record (6), Read-Multiple-Objects-Specific-Multiple-Record (7), Read-1st-Objects-Specific-Multiple-Record (8), and Read-Data-Element-List-Specific-Multiple-Record (9)

See 7.4.52 for details of what **Objects** are relevant to each **Read-Record-Type**

Max-App-Length: INTEGER (1..65535)

This only applies to Read-Type (8) and is expressed in bytes

10.26.2 Read-Multiple-Records response

The **Read-Multiple-Records** response has the following argument and field names:

Completion-Code (see 9.2)
Execution-Code (see 9.3)
Multiple-Records-Directory-Structure-List (see 11.9)
Multiple-Records-Header-Structure-List (see 11.10)
Multiple-Records-Preamble-Structure (see 11.11)
Read-Objects-Response-List (see 11.14)
Read-OIDs-Response-List (see 0)

Read-Multiple-Records response

Module-OID: OBJECT IDENTIFIER = 1 0 15961 127 33

Multiple-Records-Header-Structure-List: [Conditional] List of <Multiple-Records-Header-Structure>

Multiple-Records-Preamble-Structure: [Conditional] List of <Multiple-records>

Multiple-Records-Directory-Structure-List: [Conditional] List of <Multiple-Records-Directory-Structure>

Read-OIDs-Response-List: [Conditional] List of <Read-OIDs-Response>

Read-Objects-Response-List: [Conditional] List of <Read-Objects-Response>
Completion-Code: INTEGER

Possible Values:

Value	Definition
0	No-Error
8	Singulation-Id-Not-Found
48	Multiple-Records-Directory-Not-Present
255	Execution-Error

Additional Completion-Codes apply to the:
Read-Objects-Response-List

Execution-Code: INTEGER
Possible Values: As defined in 9.3.

10.27 Delete-Multiple-Record

10.27.1 Delete-Multiple-Record command

The **Delete-Multiple-Record** command instructs the interrogator to either mark a Multiple Record as deleted or to physically delete the record.

The **Delete-MR-Method** argument in the command instructs the interrogator as to which of these two methods is used.

If the method is mark record as deleted, the bytes that make up the record are not actually deleted, but the subject record and its entry in the directory (if present) have code values set to indicate that the record is no longer to be treated as valid. If either the record preamble or the directory is locked, then the command cannot be invoked. If the record is part of a hierarchical structure, then all the child(ren) records from the lowest level need to be deleted first. If any of these record preambles, or the associated directory entry, is locked then none of the records can be deleted.

If the method is to physically delete the record, then all the encoding in the record is changed to make this part of the memory available for future encoding. This method cannot be used if any part of the record is locked, but if the preamble of the record is not locked, then the alternate method (mark record as deleted) may be attempted.

The **Access-Password** in the command is used to match that on the RFID tag to permit writing data to the RFID tag, and deleting implies a write transaction. This only applies to the ISO/IEC 18000-6 Type C tag, and can only be applied to that RFID tag if such a password had previously been encoded.

The **Delete-Multiple-Record** command has the following arguments:

Access-Password (see 7.4.1)
Object-Identifier (see 7.3.2)
Singulation-Id (see 7.2.1)

Delete-Multiple-Record command

Module-OID: OBJECT IDENTIFIER = 1 0 15961 126 34

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

Access-Password: [OPTIONAL] BYTE STRING (4 bytes)

Delete-MR-Method: INTEGER

Possible Values:

Value	Definition
0	Mark record as deleted
1	Physically delete to record

Object-Identifier: OBJECT IDENTIFIER

This is the **Object-Identifier** to the level where the final arc is that of the instance-of or hierarchical identifier

10.27.2 Delete-Multiple-Record response

The **Read-Multiple-Records** response has the following field names:

Completion-Code (see 9.2)
Execution-Code (see 9.3)

Delete-Multiple-Record response

Module-OID: OBJECT IDENTIFIER = 1 0 15961 127 34

Completion-Code: INTEGER

Possible Values:

Value	Definition
0	No-Error
8	Singulation-Id-Not-Found
13	Object-Identifier-Not-Found
25	Password-Mismatch
49	Record-Not-Deleted-Preamble-Locked
50	Record-Not-Deleted-Directory-Locked
51	Record-Not-Deleted-Lower-Level-Preamble-Locked
52	Record-Not-Deleted-Encoding-Locked
255	Execution-Error

Execution-Code: INTEGER

Possible Values: As defined in 9.3.

11 Arguments

11.1 Add-Objects

This argument provides a list of **Object-Identifiers** and **Objects** to be written to the RFID, either to a "blank" tag or to append to data already encoded.

The **Add-Objects** argument has the following arguments:

Avoid-Duplicate (see 7.4.6)
Compact-Parameter (see 7.3.5)
Object (see 7.3.4)
Object-Identifier (see 7.3.2)
Object-Lock (see 7.3.6)

Add-Objects argument

Object-Identifier: OBJECT IDENTIFIER

Avoid-Duplicate: BOOLEAN
If TRUE, the interrogator shall check that the subject Object-Identifier is not already encoded on the RFID tag.

Object: BYTE STRING

Compact-Parameter: INTEGER (0..15)
Possible Values:

Value	Definition (see 7.3.5 for further details)
0	Application-Defined
1	Compact
2	UTF8-Data
3	Packed-Objects
4	Tag-Data-Profile

Object-Lock: BOOLEAN
If TRUE the interrogator shall lock the relevant Data-Set

If the **Compact-Parameter** is **Packed-Objects**, then this shall be the same for each component in the list; and the value of **Object-Lock** shall also be consistent throughout the list.

11.2 DSFID-Constructs

DSFID-Constructs argument

Access-Method: INTEGER

Possible Values

Value	Definition
0	No-Directory
1	Directory
2	Packed-Objects
3	Tag-Data-Profile
4	Multiple-Records
5 to 15	Reserved for extension when extensions to DSFID are implemented

Data-Format: INTEGER

Possible Values

Value	Definition
0	Not-Formatted
1	Full-Featured
2	Root-OID-Encoded
3 to 28	As published by the Registration Authority of ISO/IEC 15961-2
29	For closed systems compliant with ISO/ IEC 15962
30	For closed systems with proprietary encoding
31	Not applicable as an assigned DSFID
32 to 287	Reserved as an extension for multiple byte Data-Format

11.3 EPC-UIImemory

EPC-UII-Memory argument

Protocol-Control-Word: BYTE STRING (2)

EPC-Code: BYTE STRING

11.4 Ext-DSFID-Constructs

This argument provides a list of additional arguments that are used to indicate additional features on the tag, or processes that the Data Processor shall use to complete the encoding on the RFID tag. The argument is also used for some responses.

If either or both the **Memory-Length-Encoding** bits are = 1, the Data Processor shall calculate the relevant number of blocks and encode this on the RFID tag compliant with the Extended DSFID syntax.

If either or both the **Data-CRC-Indicator** bits are = 1, the Data Processor shall calculate the relevant data CRC and encode this in the appropriate locations in the memory.

If the application specifies that the **Battery-Assist-Indicator**, and/or **Full-Function-Sensor-Indicator**, and/or **Simple-Sensor-Indicator** are to be set = 1 (TRUE), the Data Processor shall encode these settings whether or not the selected feature is actually supported by the RFID tag.

Additional bytes may be encoded in the **Extended-DSFID** for future encoding on the RFID tag by specifying the number of **DSFID-Pad-Bytes.**

The arguments **Length-Of-Encoded-Data** and **Memory-Capacity** are returned as arguments when the **Ext-DSFID-Constructs** is included in a response.

The **Ext-DSFID-Constructs** argument has the following arguments:

Battery-Assist-Indicator (see 7.4.7)
Data-CRC-Indicator (see 7.4.11)
DSFID-Pad-Bytes (see 7.4.16)
Full-Function-Sensor-Indicator (see 7.4.20)
Length-Of-Encoded-Data (see 7.5.4)
Memory-Capacity (see 7.5.7)
Memory-Length-Encoding (see 7.4.35)
Simple-Sensor-Indicator (see 7.4.58)

Ext-DSFID-Constructs argument

Memory-Length-Encoding: BIT STRING

Possible Values

Value	Definition
00	No encoded length or memory capacity is small (i.e. not more than 256 bits)
01	Memory capacity is defined
10	The length of encoded data is defined
11	Both memory capacity and length of encoding are defined

Data-CRC-Indicator: BIT STRING

Possible Values

Value	Definition
00	No data CRC
01	Data CRC applied to each individual data set
10	Data CRC applied only to the entire encoded data
11	Data CRC applied to each data set and to the entire encoded data

Simple-Sensor-Indicator: BOOLEAN
If a simple sensor is on the tag, this is set as TRUE.

Battery-Assist-Indicator: BOOLEAN
If this is a battery assisted RFID tag, this is set as TRUE.

Full-Function-Sensor-Indicator: BOOLEAN
If a full-function sensor is on the tag, this is set as TRUE.

DSFID-Pad-Bytes: [Optional] INTEGER
This is the number of additional bytes requested by application to support additional arguments to be added at a later time, e.g. the **Memory-Capacity** and/or the **Length-Of-Encoded-Data**. In a response this might also include the number of pad bytes added to enable the extended DSFID to be locked on a lock-block boundary.

Memory-Capacity: INTEGER
This is only included in a response that included the Ext-DSFID-Constructs argument

Length-Of-Encoded-Data: INTEGER
This is only included in a response that included the Ext-DSFID-Constructs argument

11.5 ISO-UIImemory

ISO-UIImemory argument

Protocol-Control-Word: BYTE STRING (2)

DSFID: BYTE [OPTIONAL]
This is not returned when the AFI declares that that the Object is a Monomorphic-UII

Read-Objects-Response-List: List of <Read-Objects-Response>

11.6 Item-Related-Add-Objects

The command argument **Item-Related-Add-Objects** has the same function and structure as the **Add-Objects** command argument (see 11.1), except that it is encoded in the item-related segment of an RFID tag that can address multiple segments in the same air interface transactions. This specific argument enables it to be distinguished from the **UII- Add-Objects** (see 11.16).

11.7 Item-Related-DSFID-Constructs

The command argument **Item-Related-DSFID-Constructs** has the same function and structure as the **DSFID-Constructs** command argument (see 11.2), except that it is encoded in the item-related segment of an RFID tag that can address multiple segments in the same air interface transactions. This specific argument enables it to be distinguished from the **UII-DSFID-Constructs** (see 11.17).

11.8 Multiple-Records-Constructs

This argument provides a list of additional arguments that are used when writing a multiple record to the Logical Memory.

The BOOLEAN command argument **Append-To-Existing-Multiple-Record** (see 7.4.4) is used to declare, if TRUE, whether the list of data objects is to be appended to a pre-existing record, subject to all the component arguments matching fields already encoded for the multiple record. If FALSE, this command shall be used to create a new record.

The command argument **Application-Defined-Record-Capacity** (see 7.4.5) simply provides the application with the choice to set the memory capacity to a given size or to allow the Data Processor to block align the record. Allowing the Data Processor to control this is recommended unless there is a reason for the application to set a larger memory capacity, for example to enable additional data elements to be encoded at a later time. Therefore the **Record-Memory-Capacity** is only required if the application needs to control this aspect of memory.

The **Record-Type-Classification** argument (see 7.4.56) needs to be aligned with the rules defined in the MR-header. If the wrong value is used then the command shall be rejected by the Data Processor.

When constructing a command to encode a record that is part of a hierarchical structure, reference should be made to the application standard to ensure that the hierarchical relationships are correct. The **Identifier-Of-My-Parent** argument (see 7.4.22) shall be based on a record that has already been the subject of a previous write command. If there is a mismatch, then the record is not encoded.

The **Number-In-Data-Element-List** argument (see 7.4.41) only applies if the record is a data element list and is used to provide additional information to the Data Processor to ensure that the correct number of data elements with the same true **Relative-OID** is encoded. This might be particularly useful when the data varies in format and length between instances of the data element.

The arguments **Lock-Record-Preamble** (see 7.4.30), **Update-Multiple-Records-Directory** (see 7.4.62), and **Lock-Directory-Entry** (see 7.4.28) all provide instructions to the Data Processor and are generally associated with the subject record. However, if the **Update-Multiple-Records-Directory** argument is set to TRUE and the directory does not yet exist, then a complete directory shall be created. Irrespective of the setting of the argument, if a directory is already encoded, then the Data Processor shall automatically fully update the directory.

Multiple-Records-Constructs argument

Append-To-Existing-Multiple-Record BOOLEAN

If set to TRUE, the Data Processor shall check that all the arguments match with an existing multiple record and that none of the **Relative-OID** values already exist in the current version of the record. In the case that the existing record is defined as a data element list, then none of the list element number values in this command shall already be encoded in the current version of the record. If set to FALSE, the Data Processor shall encode the record as a new record.

Application-Defined-Record-Capacity: BOOLEAN

If set as TRUE, the application needs to define the Record-Memory-Capacity argument. If FALSE, the Data Processor shall simply block align the record and encode the record capacity as the minimum required to support the record.

Record-Memory-Capacity: [Optional] INTEGER

This is the total number of blocks that the application requires for the record, with any unused blocks to be set aside for additional encoding in this record in the future.

Record-Type-Classification: BIT STRING

Possible Values:

Value	Definition
000	stand-alone record, with an instance-of arc = 0
001	stand-alone record, with an instance-of arc >0
010	hierarchical record, top level
011	hierarchical record, has both parent and child(ren)
100	hierarchical record, data element list
101	other hierarchical record, no further children
110	Not relevant to this command (because it is associated with deleted records)
111	reserved

Identifier-Of-My-Parent: [Conditional] INTERGER

This value is the same as the hierarchical identifier of the parent record

Number-In-Data-Element-List: [Conditional] INTERGER (1..255)

This value is provided when the record classification is {100} to assist the data processor in encoding the associated Add-Objects argument

Lock-Record-Preamble: BOOLEAN

If this is set as TRUE, then the preamble shall be block aligned and locked

Update-Multiple-Records-Directory: BOOLEAN

If this is set as TRUE, then the directory is updated with this record, and any others not on the directory

Lock-Directory-Entry: BOOLEAN

If this is set as TRUE, then the directory entry for the record shall be block aligned and locked

11.9 Multiple-Records-Directory-Structure

This response argument provides a list of the content of the multiple records directory in a manner that the information can be used by the application to construct additional commands.

The **Multiple-Records-Directory-Structure** argument contains the following arguments and field names:

DSFID-Constructs (see 11.2)
Ext-DSFID-Constructs (see 11.4)
Hierarchical-Identifier-Arc (see 7.4.21)
Identifier-Of-My-Parent (see 7.4.22)
Instance-Of-Arc (see 7.4.25)
Record-Type-Arc (see 7.4.55)
Record-Type-Classification (see 7.4.56)
Sector-Identifier (see 7.4.57)
Start-Address-Of-Record (see 7.4.59)

Multiple-Records-Directory-Structure argument

DSFID-Constructs-list: [Conditional] List of <DSFID-Constructs>

Ext-DSFID-Constructs-list: [Conditional] List of <Ext-DSFID-Constructs>

Sector-Identifier: [Conditional] INTEGER
If the sector identifier has not been provided by the MR-header, then the value in the record preamble is valid

Record-Type-Classification: BIT STRING
Possible Values:

Value	Definition
000	stand-alone record, with an instance-of arc = 0
001	stand-alone record, with an instance-of arc >0
010	hierarchical record, top level
011	hierarchical record, has both parent and child(ren)
100	hierarchical record, data element list
101	other hierarchical record, no further children
110	Not relevant to this command (because it is associated with deleted records)
111	reserved

Record-Type-Arc: INTEGER

Instance-Of-Arc: [Conditional] INTEGER

Hierarchical-Identifier-Arc: [Conditional] INTEGER

Identifier-Of-My-Parent: [Conditional] INTERGER

This value is the same as the hierarchical identifier of the parent record

Start-Address-Of-Record: EBV-8

11.10 Multiple-Records-Header-Structure

This response argument provides a list of the content of the MR-header in a manner that the information can be used by the application to construct additional commands.

The **Multiple-Records-Header-Structure** argument contains the following arguments and field names:

DSFID-Constructs (see 11.2)
Ext-DSFID-Constructs (see 11.4)
Multiple-Records-Directory-Length (see 7.4.38)
Multiple-Records-Features-Indicator (see 7.4.39)
Number-Of-Records (see 7.4.42)
Pointer-To-Multiple-Records-Directory (see 7.4.51)
Sector-Identifier (see 7.4.57)

Multiple-Records-Header-Structure argument

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

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

Multiple-Records-Directory-Length: [Optional] EBV-8

Multiple-Records-Features-Indicator: BYTE
This is a bit map that is set in this command that determines rules for the Data Processor to follow when encoding individual records.

Sector-Identifier: INTEGER
Possible Values:

Value	Definition
0	Indicates that the sector identifier varies between records, and that the true value is only obtainable for the individual record.
≠0	Indicates the true value of the sector that applies to all records.

Pointer-To-Multiple-Records-Directory: EBV-8
This value is based on responses from the interrogator on the memory mapping of the RFID tag that is being addressed.

Number-Of-Records: EBV-8
If bit 2 of the **Multiple-Records-Features-Indicator** = 0, then the value of this field is probably invalid and shall be ignored.

11.11 Multiple-Records-Preamble-Structure

This response argument provides a list of the content of the record preamble in a manner that the information can be used by the application to construct additional commands.

The **Multiple-Records-Preamble-Structure** argument contains the following arguments and field names:

Data-Length-Of-Record (see 7.4.12)
DSFID-Constructs (see 11.2)
Encoded-Memory-Capacity (see 7.4.18)
Ext-DSFID-Constructs (see 11.4)
Hierarchical-Identifier-Arc (see 7.4.21)
Identifier-Of-My-Parent (see 7.4.22)
Instance-Of-Arc (see 7.4.25)
Record-Type-Arc (see 7.4.55)
Record-Type-Classification (see 7.4.56)
Sector-Identifier (see 7.4.57)

Multiple-Records-Preamble-Structure argument

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

Ext-DSFID-Constructs-list: [Conditional] List of <Ext-DSFID-Constructs>

Encoded-Memory-Capacity: EBV-8
This is the size of the memory that has been reserved for the record, in terms of write blocks.

Data-Length-Of-Record: [Conditional] EBV-8
This is the size of the encoded record, in terms of write blocks.

Sector-Identifier: [Conditional] INTEGER
If the sector identifier has not been provided by the MR-header, then the value in the record preamble is valid

Record-Type-Classification: BIT STRING
Possible Values:

Value	Definition
000	stand-alone record, with an instance-of arc = 0
001	stand-alone record, with an instance-of arc >0
010	hierarchical record, top level
011	hierarchical record, has both parent and child(ren)
100	hierarchical record, data element list
101	other hierarchical record, no further children
110	Not relevant to this command (because it is associated with deleted records)
111	reserved

Record-Type-Arc: INTEGER

Instance-Of-Arc: [Conditional] INTEGER

Hierarchical-Identifier-Arc: [Conditional] INTEGER

Identifier-Of-My-Parent: [Conditional] INTERGER
This value is the same as the hierarchical identifier of the parent record

11.12 Packed-Object-Constructs

The following arguments only apply to **Packed-Objects**, and are ignored if the **Access-Method** is not **Packed-Objects**.

The **Packed-Objects-Constructs** argument has the following arguments:

Block-Align-Packed-Objects (see 7.4.9)
Editable-Pointer-Size (see 7.4.17)
ID-Type (see 7.4.24)
Object-Offsets-Multiplier (see 7.4.44)
Packed-Object-Directory-Type (see 7.4.45)
PO-Directory-Size (see 7.4.48)
PO-ID-Table (see 7.5.10)
PO-Index-Length (see 7.4.49)

Packed-Objects-Constructs argument

PO-ID-Table: OCTET STRING

ID-Type: INTEGER (0..15)

Possible Values:

Value	Definition (see 7.4.24 for further details)
0	ID List
1	ID Map
2 to 15	Reserved for future definition

Packed-Object-Directory-Type: INTEGER (0..15)

Possible Values:

Value	Definition (see 7.4.45 for further details)
0	This **Packed-Object** is not a directory and does not require one
1	**Packed-Object** Presence/Absence
2	**Packed-Object** index field
3	**Packed-Object** offset
4	**Packed-Object** pointer-allocation only expecting a future command to set the directory type
5 to 15	Reserved for future definition

PO-Index-Length: INTEGER (1…7)

If this parameter is present and **Packed-Object-Directory-Type** is 2 (**Packed-Object** index field) then the implementation shall use this parameter in the POIndex Length parameter created for the POIndex Field for this Packed Object in a PO index directory. If the **Directory-Type** is not 2, this parameter shall be ignored.

Object-Offsets-Multiplier: INTEGER

If this parameter is present and **Packed-Object-Directory-Type** is 3 (**Packed-Object** offset) and **PO-Index-Length** is present, then the implementation shall use this parameter to reserve the number of bits of storage for object offsets in an AuxMap section of the directory Packed Object. If the **Directory-Type** is not 3 or a **PO-Index-Length** parameter is not present, this parameter shall be ignored. If the implementation is not able to allocate the input size number of bits for the AuxMap section of the directory Packed Object, the implementation shall return an **Insufficient-Tag-Memory** completion code.

PO-Directory-Size: INTEGER

If this parameter is present and **Packed-Object-Directory-Type** is 4 (**Packed-Object** pointer allocation only expecting a future command to set the directory type) then the implementation shall use this parameter for appropriate sizing of the (null) directory pointer created for this Packed-Object. If the **Directory-Type** is not 4, this parameter shall be ignored. If the implementation is not able to allocate the input size number of bits for the Addendum Packed Object, the implementation shall return an **Insufficient-Tag-Memory** completion code.

Block-Align-Packed-Objects: BOOLEAN

If set to TRUE, the interrogator shall ensure that this **Packed-Object** begins on a block boundary and that any necessary pad bytes are added after a previous **Packed-Object** (if present).

Editable-Pointer-Size: INTEGER

If set to a non-zero value, the interrogator shall mark the Packed Object as editable and create a pointer to an Addendum Packed Object of the size of the parameter in bits. If set to zero it indicates that the addendum subsection shall not be included.

11.13 Read-Objects

Read-Objects argument

Object-Identifier: OBJECT IDENTIFIER

Check-Duplicate: BOOLEAN
If set to TRUE, the interrogator shall check that there is only one occurrence of the OBJECT IDENTIFIER encoded on the RFID tag.

11.14 Read-Objects-Response

Read-Objects-Response argument

Object-Identifier: OBJECT IDENTIFIER

Object: BYTE STRING

Compact-Parameter: INTEGER
Possible Values:

Value	Definition
0	Application-Defined
2	UTF8-Data
14	De-Compacted-Monomorphic-UII
15	De-Compacted-Data

Lock-Status: BOOLEAN
If TRUE, the Data-Set or Packed-Object containing the Object Identifier and Object is locked.

Completion-Code: INTEGER
These definitions are supplementary to those of the response that includes this argument
Possible Values:

Value	Definition
0	No-Error
10	Duplicate-Object
13	Object-Identifier-Not-found
15	Object-Not-Read
35	Monomorphic-UII-OID-Mismatch

11.15 Read-OIDs-Response

Read-OIDs-Response argument

Object-Identifier: OBJECT IDENTIFIER

11.16 UII-Add-Objects

The command argument **UII-Add-Objects** has the same function and structure as the **Add-Objects** command argument (see 11.1), except that it is encoded in the UII segment of an RFID tag that can address multiple segments in the same air interface transactions. The argument shall only comprise on a single **Object-Identifier** and **Object**. This specific argument enables it to be distinguished from the **Item-Related-Add-Objects** argument (see 11.6).

11.17 UII-DSFID-Constructs

The command argument **UII-DSFID-Constructs** has the same function and structure as the **DSFID-Constructs** command argument (see 11.2), except that it is encoded in the UII segment of an RFID tag that can address multiple segments in the same air interface transactions. This specific argument enables it to be distinguished from the **Item-Related-DSFID-Constructs** (see 11.7).

11.18 Write-Responses

Write-Responses argument

Object-Identifier: OBJECT IDENTIFIER

Completion-Code: INTEGER
These definitions are supplementary to those of the response that includes this argument
Possible Values:

Value	Definition
0	No-Error
9	Object-Not-Added
10	Duplicate-Object
11	Object-Added-But-Not-Locked

Annex A
(informative)

Abstract syntax and transfer encoding rules

A.1 Abstract syntax

This annex is included to show the abstract syntax used in ISO/IEC 15961:2004 to support implementation of the data protocol based on that standard and ISO/IEC 15962:2004. Annex E of this part of ISO/IEC 15961 shows the original 16 commands using the abstract syntax of ISO/IEC 15961:2004. These can be used as a reference to compare with the present style of presenting the functional commands.

The abstract syntax specified in ISO/IEC 15961:2004 is based on ASN.1 as defined in ISO/IEC 8824-1. The notation shall be as specified in that standard.

A.1.1 Character set

The character set used to define an ASN.1 item shall consist of:

A to Z
a to z
0 to 9
: = , { } < . @ ( ) [ ] - ' " | & ^ * ; !

This character set is identical to that defined in ISO/IEC 8824-1.

A.1.2 Universal Types

ASN.1 supports a number of universal types that are fundamental to the syntax; sometimes called "built-in types". Each has been assigned a class tag in ISO/IEC 8824-1 to unambiguously identify each type of data. Universal types are shown in capital (uppercase) letters e.g. **UNIVERSAL**. The Universal Types used in ISO/IEC 15961:2004, together with their Class Tags, are shown in Table A.1 — Universal Types Used in ISO/IEC 15961:2004.

Table A.1 — Universal Types Used in ISO/IEC 15961:2004

Universal Type	Class Tag
BOOLEAN	1
INTEGER	2
OBJECT IDENTIFIER	6
OCTET STRING	4
RELATIVE-OID (reserved for future commands)	13
SEQUENCE & SEQUENCE OF	16

A.1.3 Type references

In addition to the Universal Types, ASN.1 enables application specific types to be defined. When a type is defined, it is given a name to reference it in another type assignment. Type references begin with an uppercase letter and the complete name is shown without spaces. There are a few variants to the presentation of the subsequent characters. ISO/IEC 15961:2004 uses the convention of mixed upper and lowercase characters e.g. **TypeReference**.

The **TypeReference** name is followed by the three character sequence ": : =" to separate it from its definition.

Examples of **TypeReference** names that are used in ISO/IEC 15961:2004 are:

ApplicationFamilyId
ObjectId
StorageFormat
TagId

All the TypeReference names used in ISO/IEC 15961:2004 are defined in the appropriate sub-clause of this Annex.

A.1.4 Element names

The components or elements of a TypeReference or enumerated list are named using a lowercase letter at the beginning, e.g. **elementName**. For some elements, further typing is required either to a TypeReference or a Universal Type.

Examples of **elementNames** that are used in ISO/IEC 15961:2004 are:

accessMethod
applicationFamilyId
applicationSubFamily
commandCode
compactParameter
object
objectId
tagId

NOTE In ISO/IEC 15961:2004, some **elementNames** and **TypeReference** names are often the same with the exception that the first letter is lowercase for the **elementName** and uppercase for the **TypeReference**.

All the **elementNames** used in ISO/IEC 15961:2004 are defined in the appropriate sub-clauses.

A.1.5 Other ASN.1 conventions illustrated

By using a simple example of ASN.1 syntax, unrelated to the purpose of, it is possible to illustrate the features of the syntax.

EXAMPLE

```
CustomerOrder : : = SEQUENCE {
    orderNumber      INTEGER
    name             OCTET STRING
    address          CustomerAddress
    productDetails SEQUENCE OF SEQUENCE {
                     productCode      OBJECT IDENTIFIER
                     quantity         INTEGER (1..999)
                     } ,
    urgency          ENUMERATED {
                     nextDay (0),
                              -- excludes Saturday and Sunday
                              firstClass (1),
                              roadTransport (2),
                              -- typically three days
                              }
        }
```

In this example:

- The double colon and equal sign : : = separates the named TypeReference **CustomerOrder** from the definition.
- The curly brackets { } following **SEQUENCE** and the end, specify that **orderNumber, name, address, productDetails** and **urgency** are all elements of the CustomerOrder type reference.
- The element name **address** is further specified in the **CustomerAddress** type reference (excluded from the example for brevity).
- The element **productDetails** consists of two further elements **productCode** and **quantity**. This pair of elements is repeated n times, based on the **SEQUENCE OF SEQUENCE**. The { } define the boundary of the Type.
- The element **urgency** offers one of three codes: 0, 1 or 2 by the use of the **ENUMERATED** type. The { } define the boundary of the Type.
- Comments in this example "excludes Saturday and Sunday" and "typically three days" are preceded by the double dash "--".
- The **INTEGER** value for the element **quantity** is constrained by the "(1..999)" to be any value in the range 1 to 999, making it an error to order 1000 or more items of a **productCode**.

A.1.6 Modular structure of ASN.1 syntax

In keeping with the ASN.1 standards, the syntax in ISO/IEC 15961:2004 is presented in a modular format. Separate modules are used for the commands and for the responses. Each module contains:

- A unique name.
- A unique Object Identifier that refers to this particular standard (in accordance with ISO/IEC 8824-1). The penultimate arc is either "commandModules (126)" or "responseModules (127)" to distinguish the data flows. The final arc of each command/response pair has the same name and value to link these together.
- The use of the key words DEFINITIONS, BEGIN and END to be compliant with ISO/IEC 8824-1 and to allow the modules to be processed through compiler tools.
- A statement that this part of ISO/IEC 15961 uses "EXPLICIT TAGS", which indicates that all of the elements ultimately encoded as UNIVERSAL TYPES.

The structure of a command module follows the following common format:

```
Module Name
  {ISO(1) standard(0) rfid-data-protocol (15961) commandModules (126) moduleName(n)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN
```

```
CommandName
-- assignments

END
```

The responseModule follows a similar format.

Within each module, all of the elements are defined in such a way as to reduce these to UNIVERSAL TYPES. This avoids the need to implement any import function within the module.

A.2 Transfer syntax

A.2.1 Structure of the transfer encoding

The structure of the transfer encoding for the data protocol for RFID for Item Management as originally specified in ISO/IEC 15961:2004 is described below:

1. type identifier octet(s) that encode the ASN.1 tag (class and number) of the Type used to qualify the data value;

2. length bytes that define a count of the bytes that make up the contents;

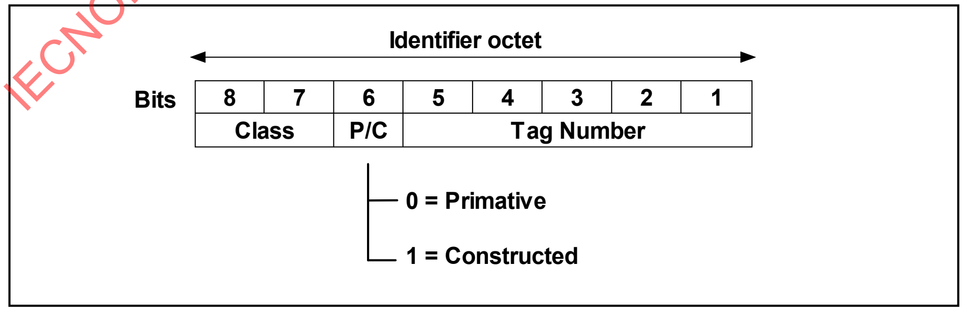
3. content (or value) bytes.

This is sometimes known as Type, Length, Value (**TLV**). When the encoding is based on a sequence of **TLV**, it is known as Primitive encoding. The value V can be a triplet of **TLV**, and when this structure is used it is known as Constructed encoding, for example **TL TLV TLV TLV**. The choice between Primitive and Constructed encoding is largely determined by the Basic Encoding Rules of ISO/IEC 8825-1. The constructor types of SEQUENCE and SEQUENCE OF that shall use a Constructed encoding to be fully compliant the ASN.1 standards. Otherwise, the rules of ISO/IEC 8825-1 require that Primitive encoding is used; or offer a choice, in which case only Primitive encoding are used in ISO/IEC 15961:2004. The option is clearly defined for each of the Universal Types that have their BER encoding rules defined in subsequent clauses.

The module OBJECT IDENTIFIER shall be encoded in a TLV structure at the beginning of the transfer

A.2.2 Encoding the ASN.1 type identifier

The ASN.1 type identifier shall be encoded as a single octet for the Types defined in ISO/IEC 15961:2004, as illustrated in Figure A.1 — Type Identifier Octet Structure.



where: bits 8 and 7 encode the class of ASN.1 tag
bit 6 encode whether this is a Primitive or Constucted tag
bits 5 to 1 encode the number of the ASN.1 tag

Figure A.1 — Type Identifier Octet Structure

The 2-bit value for Class shall be one of the values defined in Table A.2 — Encoding of ASN.1 Class of Tag. For the Universal Types defined in ISO/IEC 15961:2004, the value shall be '00_2'.

Table A.2 — Encoding of ASN.1 Class of Tag

Class	Bit 8	Bit 7
Universal	0	0
Application	0	1
Context-specific	1	0
Private	1	1

The single bit value for the 'P/C' component shall be set to '0_2' to indicate Primitive encoding structures, or shall be set to '1_2' to indicate constructed encoding structures.

The 5-bit value for the ASN.1 tag shall encode the Class tag number as a binary integer with bit 5 as the most significant bit. The Class tags specified for ISO/IEC 15961:2004 are defined in Table A.1 — Universal Types Used in ISO/IEC 15961:2004.

EXAMPLE:

Universal Type =	OBJECT IDENTIFIER
ASN.1 Type identifier =	00 0 00110

A.2.3 Length encoding

Although ISO/IEC 8825-1 allows other forms of length encoding, only the definite form shall be used in ISO/IEC 15961:2004 because this applies to both primitive and constructed encoding. For the definite form, the length bytes shall consist of one or more bytes, depending on the number of bytes in the contents. If the number of bytes in the contents is less than, or equal to, 127, then a single length octet shall be used. Bit 8 shall be '0_2' and bits 7 to 1 shall encode the number of bytes in the content (which may be zero) as an unsigned binary integer with bit 7 as the most significant bit.

EXAMPLE

L = 38 is encoded as 00100110_2

If the number of bytes in the contents is more than 127, then two or more length bytes shall be used. The length shall be converted to an octet aligned value, for example, a length of 201 bytes converts to $C9_{16}$ (or 11001001). This value is encoded in the second and subsequent bytes. The first octet shall be encoded as follows:

a. Bit 8 shall be 1_2.

b. Bits 7 to 1 shall encode the number of subsequent bytes in the length bytes, as an unsigned binary integer with bit 7 as the most significant bit.

c. The value 11111111_2 shall not be used to allow for future extension:

EXAMPLE

Length of content	=	357
Convert to HEX	=	01 65_{16}
	=	00000001_2 01100101_2

As this is 2 bytes, the first octet = 100000010_2

The complete length encoding is:

	10000010	00000001	01100101_2
=	82	01	65_{16}

A.2.4 Contents octets

The contents octets encode the data value, which can be zero, one or more octets, depending on the Universal Type as specified in subsequent subclauses.

A.2.5 Encoding of a BOOLEAN value

The encoding of a BOOLEAN value shall be primitive to comply with ISO/IEC 8825-1. The BOOLEAN value shall be encoded in a single octet. If the BOOLEAN value is FALSE, the octet shall be zero. If the BOOLEAN value is TRUE, the octet shall have any non-zero value, at the sender's option.

A.2.6 Encoding an INTEGER value

The encoding of an INTEGER value shall be primitive to comply with ISO/IEC 8825-1. The integer shall be encoded in one or more octets using the following procedures.

For positive integers and zero:

1. The whole number is converted to a binary integer number in a bit field with the most significant bit first.

2. The bit field is aligned to octet boundaries by adding leading zero bits.

3. If the high order bit is 1, add a pad octet 00_{16} as a prefix.

NOTE The high order bit of 0 denotes that the encodation is of a positive integer value.

EXAMPLE

Integer 128		
Step 1:		10000000
Step 2:		10000000
Step 3:	00000000	10000000

For negative integers, the encodation is to a twos-complement rule:

1. The whole number is converted to a binary integer number in a bit field with the most significant bit first.

2. The bit field is aligned to octet boundaries by adding leading zero bits.

3. The binary value from Step 2 is bit complemented (i.e. 0 to 1, 1 to 0).

4. The twos-complement rule is applied adding 1_2 to the bit string of Step 2.

5. If the high order bit is 0, add a pad octet FF_{16} as a prefix.

NOTE The high order bit of 1 denotes that the encodation is of a negative integer value.

EXAMPLE

Integer -27066		
Step 1:	1101001	10111010
Step 2:	01101001	10111010
Step 3:	10010110	01000101
Step 4:	10010110	01000110

For decoding, the lead bit of the encoded integer value identifies whether the value is positive or negative.

If it is a positive value, conversion takes place on the remaining bits with the least significant bit being in position 0. The decimal integer value is the sum of the values 2^n, where n is the position number, or:

$$\sum_{n=0}^{p-1} 2^n$$

If it is a negative value, conversion takes place on the remaining bits with the least significant bit being in position 0. The first stage is to create a decimal integer value as the sum of the values of 2^n. The second stage takes this as a positive decimal integer from which is subtracted the value 2^p, where p is the position number of the lead bit that identifies this as a negative integer. As an equation, this is:

$$\sum_{n=0}^{p-1} 2^n - 2^p$$

EXAMPLE

10010110	01000110		
1			indicates -ve
0010110	01000110	=	5702
$2^p = 2^{15} =$			32768
5702 - 32768 = -27066			

A.2.7 Encoding the OBJECT IDENTIFIER value

The encoding of an OBJECT IDENTIFIER value shall be primitive to comply with ISO/IEC 8825-1. The Object Identifier value is encoded as a series of octet aligned values as follows:

1. The first two arcs of the registration tree are encoded as a single integer using the formula:

   40f + s

where	f	=	the value of the first arc
	s	=	the value of the second arc

2. The value "n" of each additional arc is encoded into an octet-aligned-bit-field. This is done as follows for values of "n":

   a. For n < 128:

      the decimal value is converted to binary and encoded in a single octet; thus bit 8 is set to zero

   b. For : $128 \leq n < 16384$:

      the decimal value is converted to binary and subdivided into two 7-bit strings: bit 7 to bit 1, bit 14 to bit 8. Each of these new bit strings is encoded in an octet; with bit 8 of the first octet set to 1, bit 8 of the last octet set to 0.

c. For n ≥ 16384:

the decimal value is converted to binary and subdivided into 7-bit strings: bit 7 to bit 1, bit 14 to bit 8, bit 21 to bit 15, and so on. Each of these new bit strings is encoded in an octet; with bit 8 of the first octet set to 1, bit 8 of the last octet set to 0 and bit 8 of the intervening octet(s) set to 1. The example below shows the process.

EXAMPLE

1. Value = 91234_{10}

   = $1\ 01100100\ 01100010_2$

2. Split into 7-bit strings

   0000101 1001000 1100010

3. Add prefix bits 0 for last octet

   1 for preceding octet(s)

   10000101 11001000 01100010

Using this technique, the length of each component arc of the OBJECT IDENTIFIER is self declaring. The first octet always defines the first two arcs. Each subsequent arc is defined by one octet if the lead bit of the next octet is 0; and multiple octets if the lead bit is 1, the group of octets ends with the octet with its lead bit equal 0. The arc value is encoded in the sequence of 7-bit values.

EXAMPLE

[00101000]	1[1111000]	0[1001010]	0[0000001]
(1 x 40) + 0	15434		1
1 0	15434		1

Although the number of arcs allows for an OBJECT IDENTIFIER of any length, ISO/IEC 15961:2004 limits the length of the encoded value to be no more than 127 octets. This is a constraint placed to meet encoding requirements on the RF tag and the structure of the Logical Memory.

NOTE The constraint is on the encoded length of the OBJECT IDENTIFIER and not the number of arcs. It should also be understood that an OBJECT IDENTIFIER encoded in 127 octets is highly unlikely.

A.2.8 Encoding an OCTET STRING value

Although the Basic Encoding Rules of ISO/IEC 8825-1 permit both forms of encoding, ISO/IEC 15961:2004 only supports primitive encoding of an OCTET STRING value.

The primitive encoding contains zero, one or more octets equal in value to the octets in the application data value. The encoded octets appear in the same order as they appear in the data value and with the most significant bit of an octet being aligned in both the encoded and data presentations.

NOTE This means that, for open systems, the octet sequence and bit ordering shall be output by the receiving system exactly as input by the sending system. It is the responsibility of the application standards to define the requirements for the sequence.

A.2.9 Encoding a SEQUENCE value

The encoding of a SEQUENCE value shall be constructed to comply with ISO/IEC 8825-1. The contents octets shall consist of the complete TLV encoding of one data value from each of the Types listed in the

ASN.1 definition of the particular SEQUENCE Type. The data values shall be in the order of their appearance in the definition. Although ISO/IEC 8825-1 allows optional rules for Types with the keywords 'OPTIONAL' or 'DEFAULT' in the ASN.1 definition, ISO/IEC 15961:2004 requires all Types in the SEQUENCE to appear in the constructed encoding.

EXAMPLE

ASN.1 definition
SEQUENCE {orderNumber OCTET STRING, product OCTET STRING, quantity INTEGER}

with the values:
{orderNumber "ABC1234", product "widget", quantity "12"}

is encoded as:

T = SEQUENCE	L
30_{16}	14_{16}

T =		L	V
T =	OCTET STRING	L	V
	04_{16}	07_{16}	"ABC1234"
T =	OCTET STRING	L	V
	04_{16}	06_{16}	"widget"
T =	INTEGER	L	V
	02_{16}	01_{16}	$0C_{16}$

A.2.10 Encoding a SEQUENCE OF value

The SEQUENCE OF type has the same ASN.1 tag (UNIVERSAL 16) as the SEQUENCE type; therefore, adopting the same encoding rules. The encoding of a SEQUENCE OF value shall be constructed to comply with ISO/IEC 8825-1. The contents octets shall consist of the complete TLV encoding of each value, including the encoding of the repeated UNIVERSAL class tag of the encoded elements.

EXAMPLE

ASN.1 definition
sequence of {productCode OCTET STRING}

with the three values:
{productCode "ABC1234", "X6789Y", "PQR12345"}

is encoded as:

T = SEQUENCE	L
30_{16}	$1B_{16}$

T =	OCTET STRING	L	V
	04_{16}	07_{16}	"ABC1234"
T =	OCTET STRING	L	V
	04_{16}	06_{16}	"X6789Y"
T =	OCTET STRING	L	V
	04_{16}	08_{16}	"PQR12345"

Annex B
(informative)

Accommodating established data formats

This part of ISO/IEC 15961 has been prepared on the basis that its Object-based protocol differs from the message based protocols and syntax of some AIDC application standards. Therefore, basic data Objects need to be presented in a manner relevant to the application standard, for example in terms of:

— the data **Object** being supported in the data dictionary

— the format of the data (e.g. numeric, alphanumeric), including its length

— combinations of data Objects which are valid or illegal

These features are outside the scope of this part of ISO/IEC 15961 and are the responsibility of the application.

Some conversion process is necessary until the application systems can handle data and identifiers in the format specified in this part of ISO/IEC 15961. It is possible to have two independent implementation paths: one to write data, and one to read data.

For some major applications, some rigorous rules exist of what constitutes legitimate data. Software exists to ensure compliance to this format when using the existing message based syntax. Users need to ensure that, as they implement an Object based method of writing data, the data itself follows the basic rules. Figure B.1 — Data Flow Model: Prepare Data of Existing Application Standards illustrates this schematically.

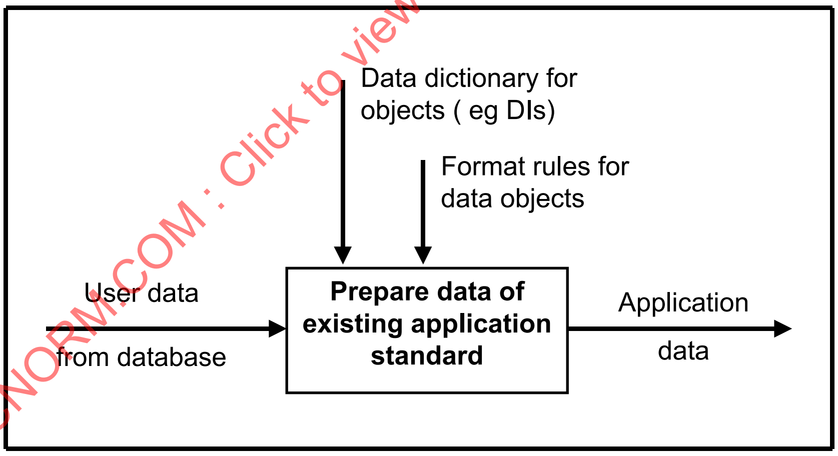

Data dictionary for objects (eg DIs)
Format rules for data objects
User data from database
Prepare data of existing application standard
Application data

Figure B.1 — Data Flow Model: Prepare Data of Existing Application Standards

A similar process is required when data is read from an RFID tag.

AIDC technologies based on write-once-read-many-times (WORM) technology can rely on the fact that data, as written, is what is read. This means the message syntax is encoded in the data carrier. The read-write capabilities of RFID, and the Object based nature of the data protocol of this part of ISO/IEC 15961, means that an established syntax has to be constructed based on the Object Identifier structure. While there is a requirement to output data with a particular data syntax (e.g. that of ISO/IEC 15434), then a conversion module is required to correctly map the set of Object Identifiers and data Objects to the message format required for the application of this part of ISO/IEC 15961. This shall require the inversion of the conversion rules for some of the application data.

In addition, the message syntax of the established application standard needs to be created. This process generally requires all but the final arc of the Object Identifier to be discarded and for data separators (compliant with the application standard) to be inserted correctly. Refer to the appropriate application standards for the precise rules.

A logical development step is for the application standard to develop procedures to accept output based on transfer syntax.

Annex C
(informative)

Relating data Objects

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

With an Object-based system (such as the Data Protocol of this part of ISO/IEC 15961 and ISO/IEC 15962) operating at a base level, there is a risk of creating false links (i.e. product code A could be linked to quantity of product B). The problem can be overcome using one of the techniques described below. The methods should only be adopted if incorporated into the application standard associated with the item being managed. The illustrations limit the number of constructed data elements to 255 per RFID tag, but different rules could be developed if a greater number of constructions is required. Either rule is transparent to the complete Data Protocol of this part of ISO/IEC 15961 and ISO/IEC 15962, and so requires the processing to be implemented as part of the application. The options are included in this part of ISO/IEC 15961 to describe robust ways to preserve an Object-based data capture process using the Object Identifier tree structure.

C.1 Concatenation technique

Specific Object Identifiers can be created that link a defined set of attributes in a concatenated manner.

EXAMPLE

lowest arc 245 =

— sequence number	1 octet
— product code	8 octets
— quantity	1 octet
— batch number	4 octets

In this case the first byte of the **Object**, the sequence number, distinguishes one similar Object data from another.

Using this technique, each different arrangement of basic elements to create the concatenated construction would be given a different final node. So, the concatenation of product + quantity + expiry date would have its lowest arc value different from that for product + quantity + batch.

This method is more suitable when fixed combinations of element have to be created and the length of each **Object** is fixed.

C.2 Object identifier extension technique

The basic Object Identifier can be extended by the addition of a new final arc, with this as a 'linking' value.

EXAMPLE

The following three elements are to be linked:

— product code - final arc	48
— quantity - final arc	17
— batch - final arc	20

Assume that there are two different products whose details are encoded on the RFID tag. So the linking extensions 1 and 2 apply. Six individual **Object-Identifiers** are encoded:

```
...   48 1
....  48 2
....  17 1
....  17 2
....  20 1
....  20 2
```

The extension value is used to link the different Objects as a logical combination.

This method is more suitable when many different combinations of element have to be created and the length of, at least, one Object can vary between occurrences.

The Extension technique for relating Objects and their associated physical entities is similar to Scheme B for applying data security (see Annex C.1). Therefore, for any one **Object-Identifier**, the technique shall only be applied to data security or to linking physical entities.

Annex D
(informative)

Data security issues

Although data security is beyond the scope of this part of ISO/IEC 15961 and ISO/IEC 15962, the following advice is provided to show how features of the Data Protocol may be used to achieve more secure data.

D.1 Object-Identifier issues

Encrypted data shall be associated with its own unique **Object-Identifier**. This ensures that authorised users can recognise encrypted data, but does not declare to other users this fact. The **Object** itself simply appears with the **Compact-Parameter** set as **Application-Defined** (see 7.3.5).

One method, called Scheme A for later reference, of creating the **Object-Identifier** is for this to have a final arc at the same level of all other final arcs in the application system. This is a systems level adoption of data security and requires all authorised users to know that the data is encrypted; however, the rules do not need to be publicly declared.

Another method, called Scheme B for later reference, for creating a unique **Object-Identifier** to identify the encrypted data is to extend the **Object-Identifier** of the plain (unencrypted) data and add an additional lower arc. This technique can be adopted bi-laterally between sender and authorised user(s), or at the systems levels for all authorised users. This technique can also be used to define the encryption type, selected keys and so on.

EXAMPLE

0 1 15961 nn nn	Plain Object
0 1 15961 nn nn 1	Encrypted Object
0 1 15961 nn nn 2	Encryption type
0 1 15961 nn nn 3	Key

Scheme B is similar to that proposed for relating Objects and their associated physical entities (see Annex C.2). Therefore, for any one **Object-Identifier**, the technique shall only be applied to data security or to linking physical entities.

D.2 The data Object

The **Object** containing the application data shall have its **Compact-Parameter** set to **Application-Defined** after encryption.

The basic **Object** should be expanded to include a pre-defined data field or signature of the authorised writing party. This will help ensure data integrity as any unauthorised modification of the encrypted **Object** without access to the private key will most probably destroy the authorised signature. This will help identify that the **Object** has been changed without authority.

If a completely different **Object-Identifier** is assigned to the encrypted data (Scheme A above), then it may need to be expanded to also contain additional unencrypted octets that define the encryption scheme and/or selection from a set of keys.

D.3 Using the Tag ID

It is possible to use the unique Tag ID, as defined for various types in ISO/IEC 18000, as a component in a secure system. The Tag ID is intended to be unique to the RFID tag, distinguishing it from all others. It is usually created at an early stage of manufacturing the RFID tag using more robust techniques than can be used to write data into the Logical Memory Map. As such, it can be used to enhance data validity. Because of potential confusion with the **Singulation-Id** of the data protocol (which can use the Tag-ID), the Tag ID incorporated into the integrated circuit is referred to as the 18000 Tag ID, for the remainder of this annex.

Where the 18000 Tag ID also acts as the **Singulation-Id** as part of the systems information, the 18000 Tag ID is provided at an early stage of communication with the RFID tag. Where the 18000 Tag ID is not provided as part of the systems information, additional commands are required to read it.

One method is to concatenate the 18000 Tag ID value to the basic **Object** data and encrypt the entire expanded **Object**. When decrypted by an authorised user, the 18000 Tag ID within the expanded **Object** can be compared with the real 18000 Tag ID to verify that they are identical. This may be applied to either Scheme A or B for creating the **Object-Identifier**, described above.

Another method is to use the 18000 Tag ID to modify the original key for encryption and decryption. This can be made to work for binary keys such as DES, where the 18000 Tag ID can be *exclusive or-ed* with the original key.

Before this approach is used, the implementers should verify that this type of modification of the key does not undermine the encryption method.

D.4 Advice on public key methods of encryption

Public key algorithms require longer keys to provide the same strength as symmetric keys. For example, a 512-bit public-key encryption cipher would have the equivalent strength of a 64-bit symmetric key cipher. This cipher length could preclude the use of public key ciphers in smaller capacity tags.

If a public key method of encryption is used, the data security will be compromised if the private key is used to encrypt the data and a public key is used to decrypt the data. Likewise, data integrity will be compromised if the public key is used to encrypt the data and the private key is used to decrypt the data. Double encryption or other means must be used to ensure data security and integrity.

The public key should not be encoded in the tag unless it is locked as an unauthorised party could violate the data integrity by overwriting the public key with another key and using another corresponding private key to encrypt altered data in the Object.

Annex E
(informative)

Original commands and responses using ASN.1 abstract syntax

The following sub-clauses show the original 16 modules using the ASN.1 abstract syntax. Cross-references, sometimes with a name change, are provided to the normative clauses that now replace those that were in the original modules.

E.1 ConfigureAfiModules

The ConfigureAfiModules consist of a commandModule, and the associated responseModule, that instruct the interrogator to write the **applicationFamilyId** (including the sub-family) into the RFID tag. A fundamental requirement of this command is that only one RFID tag shall be programmed per command. This is to ensure that the configuration process is robust, particularly in environments where more than one type of RFID tag can be present.

The abstract syntax for ConfigureAfiModules is given in Table E.1 — ConfigureAfiModules.

Table E.1 — ConfigureAfiModules

```
-- Configure AFI
-- The ConfigureAfiCommand instructs the interrogator to write the AFI (Application
-- Family Identifier, including the sub-family) into the RFID tag.  The interrogator shall
-- lock the AFI if the Lock flag is set to true.


ConfigureAfiCommand
{iso(1) standard(0) rfid-data-protocol(15961) commandModules(126) configureAfi(1)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

ConfigureAfiCommand ::= SEQUENCE {
    tagId                 OCTET STRING(SIZE(0..255)),
                         -- See Clause 7.2.1 (now renamed Singulation-Id) for detailed
                         -- specification.  TagId shall be provided by the Tag Driver for the
                         -- purposes of identifying the RFID tag unambiguously
                         -- for at least the period of a transaction.
    applicationFamilyId   ApplicationFamilyId,
    afiLock               BOOLEAN
                          -- If set to TRUE, the interrogator shall lock the AFI
    }

ApplicationFamilyId ::= SEQUENCE {
    applicationFamily     INTEGER {
                          all(0), -- address all families
                          -- values 1 - 8 reserved for definition by SC17
                          afiBlock9(9),
                          afiBlockA(10),
                          afiBlockB(11),
                          afiBlockC(12)
                          -- values 9 to 12 defined as per Annex B of this
                          -- International standard
```

```
                             -- values 13 to 15 reserved for definition by ISO/IEC
                             } (0..15),
    applicationSubFamily     INTEGER {
                             all(0),-- This value shall not be encoded in the RFID tag, and
                         -- shall only be used in a command to signal that the
                         -- interrogator shall address all subfamilies within the
                         -- selected family.
                         -- NOTE: This has little utility for this Data Protocol, but
                         -- is retained for compatibility with SC17 smart card commands
                             asf1-annex (1), -- values 1 to 15, for applicationFamily 9
                             -- to 12,defined as per Annex B of this part of ISO/IEC 15961
                             asf2-annex (2),
                             asf3-annex (3),
                             asf4-annex (4),
                             asf5-annex (5),
                             asf6-annex (6),
                             asf7-annex (7),
                             asf8-annex (8),
                             asf9-annex (9),
                             asfA-annex (10),
                             asfB-annex (11),
                             asfC-annex (12),
                             asfD-annex (13),
                             asfE-annex (14),
                             asfF-annex (15)
                             } (0..15)

-- ApplicationFamilyId is stored as a single OCTET within system information on the tag.
-- ApplicationFamilyId allows tags to be grouped according to specific families and
-- allows any such family of tags to be selectively addressed by the application. RFID tag
-- vendors may implement mechanisms in the Tag Driver and air interface specifically for
-- selective addressing of RFID tags by ApplicationFamilyId.

}
END


ConfigureAfiResponse
    {iso(1) standard(0) rfid-data-protocol(15961)responseModules(127) configureAfi(1)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

ConfigureAfiResponse ::= SEQUENCE {
    completionCode      INTEGER {
                        noError(0),
                        afiNotConfigured(1),
                        afiNotConfiguredLocked(2),
                        afiConfiguredLockFailed(3),
                        tagIdNotFound(8),
                        executionError(255)
                        },
    executionCode       INTEGER
                        -- See Clause 9.3 and notes in this syntax for a full list of
                        -- executionCodes
}
END
```

The following elementNames used in these modules are defined elsewhere in this part of ISO/IEC 15961, as detailed:

afiLock (see7.4.3)
applicationFamily (see 7.2.2)
applicationFamilyId (see 7.2.2)
applicationSubFamily (see 7.2.2)
completionCode (see 9.2)
executionCode (see 9.3)
tagId (see 7.2.1)

E.2 ConfigureStorageFormatModules

The ConfigureStorageFormatModule consists of a commandModule, and the associated responseModule, that instruct the interrogator to write the **storageFormat** (**accessMethod** and **dataFormat**) into the RFID tag. The command also instructs the interrogator to initialise the RFID tag Logical Memory Map by erasing any data already stored there. A fundamental requirement of this command is that only one RFID tag shall be programmed per command. This is to ensure that the configuration process is robust, particularly in environments where more than one type of RFID tag can be present.

If the **accessMethod** (incorporated in **storageFormat**) is specified as directory, the interrogator shall create the initial directory structure.

The ASN.1 Abstract Syntax for ConfigureStorageFormatModules is given in Table E.2 — ConfigureStorageFormatModules.

Table E.2 — ConfigureStorageFormatModules

```
-- Configure StorageFormat
-- The ConfigureStorageFormatCommand instructs the interrogator to write the
-- StorageFormat into the RFID tag, and to initialise the tag logical memory
-- map. The interrogator shall erase all the application memory, and if the
-- directory format is specified by the StorageFormat, it shall create the
-- initial directory structure. The interrogator shall lock the
-- StorageFormat if the Lock flag is set to true

ConfigureStorageFormatCommand
{iso(1) standard(0) rfid-data-protocol(15961) commandModules(126) configureStorageFormat(2)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

ConfigureStorageFormatCommand ::= SEQUENCE {
    tagId             OCTET STRING(SIZE(0..255)),
                      -- See Clause 7.2.1 (now renamed Singulation-Id) for detailed
                      -- specification.  TagId shall be provided by the Tag Driver for the
                      -- purposes of identifying the RFID tag unambiguously
                      -- for at least the period of a transaction.
    storageFormat     StorageFormat,
    storageFormatLock BOOLEAN
                      -- If set to TRUE, the interrogator shall lock the
                      -- StorageFormat
    }

StorageFormat ::= SEQUENCE {
    accessMethod   INTEGER {
                      noDirectory(0),
                      directory(1),
                            selfMappingTag(2)   -- Access to objects is via high
                            -- level commands to the RFID tag and the internal
                            -- structure of the memory inside the RFID tag is not
                            -- defined
                            } (0..3),
    dataFormat      INTEGER {
                            notFormatted(0),    -- Not formatted according
                                -- to this part of ISO/IEC 15961
                            fullFeatured(1),    -- Supports any type of
                                -- data based on full OID
                            rootOidEncoded(2),    -- Supports any type of
                                -- data with a common root-OID
                            iso15434(3),    -- root-OID is defined as
                                -- {1 0 15434}
                            iso6523(4),    -- Supports data belonging to one or
                                -- more International Code Designators compliant
                                -- with ISO/IEC 6523-1, root-OID is defined as
                                -- (1 0 6523)
                            iso15459(5),    -- Supports unique item identifiers
                                -- compliant with ISO/IEC 15459, root-OID is
                                -- defined as (1 0 15459)
                            iso15961Combined(8),    -- Supports combinations of
                                -- formats of ISO/IEC 15961, root-OID is defined
                                -- as {1 0 15961}
                            ean-ucc(9),    -- Supports data of the EAN-UCC system,
                                -- root-OID is defined as {1 0 15961 9}
                            di(10),    -- Supports Data Identifiers (as referred to
                                -- in ISO/IEC 15418), root-OID is implied to be
                                -- {1 0 15961 10}
                            iata(12)    -- Supports IATA baggage handling data elements,
                                -- root-OID is defined as {1 0 15961 11}
                            } (0..31)
}
END

ConfigureStorageFormatResponse
{iso(1) standard(0) rfid-data-protocol(15961) responseModules(127) configureStorageFormat(2)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

ConfigureStorageFormatResponse ::= SEQUENCE {
    completionCode INTEGER {
                            noError(0),
                            storageFormatNotConfigured(4),
                            storageFormatNotConfiguredLocked(5),
                            storageFormatConfiguredLockFailed(6),
                            tagIdNotFound(8),
                            executionError(255)
                            },
    executionCode   INTEGER
                            -- See Clause 9.3 and notes in this syntax
                            -- for a full list of executionCodes
}
END
```